TELECOMUNICACIÓN

Campus Sur POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Clasificación de tráfico de Internet mediante análisis de datos

AUTOR: Luis Gil Delgado

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR: Ana Belén García Hernando

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Margarita Martínez Núñez

VOCAL: Ana Belén García Hernando

SECRETARIO: Mary Luz Mouronte López

Fecha de lectura:

Calificación:

El Secretario,

Resumen

Poder clasificar de manera precisa la aplicación o programa del que provienen los flujos que conforman el tráfico de uso de Internet dentro de una red permite tanto a empresas como a organismos una útil herramienta de gestión de los recursos de sus redes, así como la posibilidad de establecer políticas de prohibición o priorización de tráfico específico.

La proliferación de nuevas aplicaciones y de nuevas técnicas han dificultado el uso de valores conocidos (*well-known*) en puertos de aplicaciones proporcionados por la IANA (*Internet Assigned Numbers Authority*) para la detección de dichas aplicaciones. Las redes P2P (*Peer to Peer*), el uso de puertos no conocidos o aleatorios, y el enmascaramiento de tráfico de muchas aplicaciones en tráfico HTTP y HTTPS con el fin de atravesar *firewalls* y *NATs* (*Network Address Translation*), entre otros, crea la necesidad de nuevos métodos de detección de tráfico.

El objetivo de este estudio es desarrollar una serie de prácticas que permitan realizar dicha tarea a través de técnicas que están más allá de la observación de puertos y otros valores conocidos. Existen una serie de metodologías como *Deep Packet Inspection* (*DPI*) que se basa en la búsqueda de firmas, *signatures*, en base a patrones creados por el contenido de los paquetes, incluido el *payload*, que caracterizan cada aplicación. Otras basadas en el aprendizaje automático de parámetros de los flujos, *Machine Learning*, que permite determinar mediante análisis estadísticos a qué aplicación pueden pertenecer dichos flujos y, por último, técnicas de carácter más heurístico basadas en la intuición o el conocimiento propio sobre tráfico de red.

En concreto, se propone el uso de alguna de las técnicas anteriormente comentadas en conjunto con técnicas de minería de datos como son el Análisis de Componentes Principales (PCA por sus siglas en inglés) y *Clustering* de estadísticos extraídos de los flujos procedentes de ficheros de tráfico de red. Esto implicará la configuración de diversos parámetros que precisarán de un proceso iterativo de prueba y error que permita dar con una clasificación del tráfico fiable. El resultado ideal sería aquel en el que se pudiera identificar cada aplicación presente en el tráfico en un clúster distinto, o en clusters que agrupen grupos de aplicaciones de similar naturaleza.

Para ello, se crearán capturas de tráfico dentro de un entorno controlado e identificando cada tráfico con su aplicación correspondiente, a continuación se extraerán los flujos de dichas capturas. Tras esto, parámetros determinados de los paquetes pertenecientes a dichos flujos serán obtenidos, como por ejemplo la fecha y hora de llagada o la longitud en octetos del paquete IP. Estos parámetros serán cargados en una base de datos *MySQL* y serán usados para obtener estadísticos que ayuden, en un siguiente paso, a realizar una clasificación de los flujos mediante minería de datos. Concretamente, se usarán las técnicas de PCA y *clustering* haciendo uso del software *RapidMiner*.

Por último, los resultados obtenidos serán plasmados en una matriz de confusión que nos permitirá que sean valorados correctamente.

Abstract

Being able to classify the applications that generate the traffic flows in an Internet network allows companies and organisms to implement efficient resource management policies such as prohibition of specific applications or prioritization of certain application traffic, looking for an optimization of the available bandwidth.

The proliferation of new applications and new technics in the last years has made it more difficult to use *well-known* values assigned by the IANA (*Internet Assigned Numbers Authority*), like UDP and TCP ports, to identify the traffic. Also, P2P networks and data encapsulation over HTTP and HTTPS traffic has increased the necessity to improve these traffic analysis technics.

The aim of this project is to develop a number of techniques that make us able to classify the traffic with more than the simple observation of the *well-known* ports. There are some proposals that have been created to cover this necessity; *Deep Packet Inspection* (DPI) tries to find *signatures* in the packets reading the information contained in them, the *payload*, looking for patterns that can be used to characterize the applications to which that traffic belongs; *Machine Learning* procedures work with statistical analysis of the flows, trying to generate an automatic process that learns from those statistical parameters and calculate the likelihood of a flow pertaining to a certain application; *Heuristic Techniques*, finally, are based in the intuition or the knowledge of the researcher himself about the traffic being analyzed that can help him to characterize the traffic.

Specifically, the use of some of the techniques previously mentioned in combination with data mining technics such as *Principal Component Analysis* (PCA) and *Clustering* (grouping) of the flows extracted from network traffic captures are proposed. An iterative process based in success and failure will be needed to configure these data mining techniques looking for a reliable traffic classification. The perfect result would be the one in which the traffic flows of each application is grouped correctly in each cluster or in clusters that contain group of applications of similar nature.

To do this, network traffic captures will be created in a controlled environment in which every capture is classified and known to pertain to a specific application. Then, for each capture, all the flows will be extracted. These flows will be used to extract from them information such as date and arrival time or the IP *length* of the packets inside them. This information will be then loaded to a MySQL database where all the packets defining a flow will be classified and also, each flow will be assigned to its specific application. All the information obtained from the packets will be used to generate statistical parameters in order to describe each flow in the best possible way. After that, data mining techniques previously mentioned (PCA and *Clustering*) will be used on these parameters making use of the software *RapidMiner*.

Finally, the results obtained from the data mining will be compared with the real classification of the flows that can be obtained from the database. A Confusion Matrix will be used for the comparison, letting us measure the veracity of the developed classification process.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

LISTA DE ACRÓNIMOS

CSV: Comma-separated values
DPI: Deep Packet Inspection
FTP: File Transfer Protocol
GPL: GNU General Public License
HTTP: Hypertext Transfer Protocol
HTTPS: Hypertext Transfer Protocol Secure
IANA: Internet Assigned Numbers Authority
IP: Internet Protocol
KDD: Knowledge Discovery in Databases
KNIME: Knstanz Information Miner
NAT: Network Address Translation
P2P: Peer to Peer
PCA: Principal Component Analysis
SQL: Structured Query Language
TCP: Transmission Control Protocol
UDP: User Datagram Protocol
WEKA: Waikato Environment for Knowledge Analysis
XML: Extensible Markup Language

1. Introducción y objetivos

Desde hace años, y cada vez más, la capacidad de conocer qué tipo de tráfico recorre las redes tanto de una empresa como de una organización es considerado de una utilidad fundamental. Se puede pensar que este tipo de capacidad sería sólo interesante para operadoras de telecomunicación, empresas dedicadas a prestar servicios en Internet, o demás empresas dedicadas a prestar servicios tecnológicos. Sin embargo cualquier organización o empresa que cuente con una red de Internet para uso de sus empleados, algo que hoy en día es bastante común, disponer de dichas herramientas es realmente útil. Realizar una mejor planificación y gestión de recursos de red, o establecer una política de prohibición y permiso de uso de ciertas aplicaciones que pueden consumir de forma excesiva los recursos disponibles son ejemplos de casos de uso.

La observación y análisis de tráfico de red es algo que ha acompañado, desde sus inicios, a Internet. Al principio, y gracias a políticas de normalización como la asignación de puertos *well-known* por parte de organismos como la IANA, era relativamente fácil clasificar los flujos de datos de cualquier red, al menos, de forma genérica, entre las distintas aplicaciones existentes. Sin embargo, mientras que las aplicaciones han evolucionado de forma considerable, el análisis de tráfico no ha avanzado tanto. El uso de puertos no conocidos e incluso aleatorios por parte de nuevas aplicaciones, junto con uso de redes P2P, y otras técnicas como enmascaramiento de paquetes dentro de tráfico web HTTP y HTTPS son muy utilizadas hoy en día y que, debido a su naturaleza, no permiten clasificar el tráfico de una forma tan simple como sólo “leyendo puertos”.

Como respuesta a dicha evolución, nuevas técnicas han aparecido para ser usadas en conjunto con las ya mencionadas: *Deep Packet Inspection* propone encontrar patrones dentro de los paquetes usando la información útil contenida en ellos, de forma que dichos patrones creen “firmas” que permitan identificarlos; *Machine Learning*, basado en parámetros estadísticos, busca una forma de predecir a qué aplicación puede pertenecer un paquete mediante un aprendizaje que puede ser automatizado y que se basa en un estudio reiterativo y acumulativo de los datos; y, por último, técnicas de naturaleza heurística, basadas en el conocimiento que puede tener el investigador tanto en tráfico de Internet en general como de la red en cuestión que esté siendo analizada.

Existe un gran número de estudios académicos sobre la aplicación de estas técnicas, como por ejemplo en [1], donde se proponen algoritmos para implementar DPI, o [2] y [3], donde se estudian distintas formas de clasificar tráfico a través de *Machine Learning*. Además, técnicas como el *clustering* y el Análisis de Componentes Principales (*PCA*), son usadas en [4] y [5] y ampliamente documentadas en [6]. Muchas de las técnicas usadas en este proyecto han sido desarrolladas mediante la lectura de [6], donde se documenta el uso de PCA en conjunto con algoritmos de clustering como *k-means* y técnicas de normalización de datos como *z-scoring*.

El objetivo de este proyecto es el desarrollo de una clasificación de tráfico de Internet mediante el uso del Análisis de Componentes Principales (PCA) y técnicas de *clustering* que permitan diferenciar entre flujos generados por distintas aplicaciones y hacer posible la clasificación de dicho tráfico.

Para ello, se generarán capturas de tráfico en un entorno controlado de forma que cada captura esté identificada con su aplicación correspondiente, los paquetes de dichas capturas serán agrupados en flujos, entendiendo un flujo como el conjunto de paquetes que cuentan con los siguientes parámetros en común: *Dirección IP Origen, Dirección IP Destino, Puerto Origen, Puerto Destino*, y tráfico *UDP* o *TCP*. Una vez obtenidos los flujos, se extraerán de ellos parámetros que serán usados para calcular estadísticos mediante su procesado en una base de datos *MySQL*. Estos estadísticos serán los usados en el proceso de minería de datos desarrollado en *RapidMiner* que agrupará los flujos en *clusters* que se corresponderán con las aplicaciones iniciales.

Este agrupamiento será entonces comparado con la clasificación de flujos real, la cual conocemos, a través de una Matriz de Confusión que nos ayudará a evaluar la validez de los resultados.

A continuación se resume la estructura del resto de esta memoria. En el siguiente capítulo, el capítulo 2, se hablará de algunas las técnicas mencionadas relativas al análisis de tráfico de Internet y a la Minería de Datos, comentando su utilidad y relación con este proyecto. A continuación, en el capítulo 3, se describe todo el proceso de obtención, extracción y procesado de datos desarrollado en este proyecto, así como la obtención y comparación de resultados. En el capítulo 4 se describen y discuten las pruebas realizadas y los resultados obtenidos. Por último, en el capítulo 5, se hablará sobre las conclusiones obtenidas y los posibles trabajos futuros con los que este proyecto podría avanzar.

2. Descripción de las tecnologías utilizadas en el proyecto

2.1. Minería de Datos

2.1.1. Concepto

La minería de datos (*Data Mining*) forma parte de la metodología *KDD* (*Knowledge Discovery in Databases*) dentro de la cual efectúa la fase del análisis. Su función principal es intentar descubrir patrones dentro de grandes conjuntos de datos que permitan solucionar problemas o simplemente, actúen como información útil a la hora de toma de decisiones o definición de líneas de actuación. Técnicas y ciencias como la inteligencia artificial, el aprendizaje automático, la estadística y las matemáticas son las bases de estos procesos realizados sobre grandes volúmenes de datos que suelen estar alojados en bases de datos y que precisan de una buena gestión y preparación antes de ser tratados.

Se trata de un proceso automático o semiautomático en el que se busca la extracción de patrones desconocidos como pueden ser la extracción de grupos entre los datos (clusters), la detección de elementos extraños (anomalías) o reglas de dependencia entre elementos. Al tratarse de grandes volúmenes de datos, pueden existir elementos relevantes que debido a su antigüedad se encuentran menos accesibles que los más actuales y, gracias a la minería de datos, es posible detectar patrones o relaciones que pueden existir entre estos elementos y los más actuales.

En general, un proceso de minería de datos conlleva las siguientes etapas:

- Selección del conjunto de datos
- Análisis de propiedades de los datos
- Transformación del conjunto de datos de entrada: se trata de la preparación óptima de los datos acorde con el tipo de técnica que se va a usar, también llamado *preprocesamiento* de los datos.
- Selección y aplicación de la técnica de minería de datos
- Extracción de resultados
- Interpretación y evaluación

Otro aspecto importante a tener en cuenta en la minería de datos es el tipo de datos con el que se trabaja, una mala preparación de estos datos, así como una mala conversión entre tipos, puede no sólo llevar a resultados inesperados o erróneos, sino también a diseños de modelos erróneos o no válidos.

En un primer nivel, los datos pueden ser divididos en categóricos y numéricos. Los numéricos son aquellas variables cuyo valor son puramente números, ya sean reales o enteros, cuyo rango puede variar. Por otro lado, los categóricos son aquellos valores que dividen los elementos en grupos, como pueden ser el sexo, la edad, la raza si hablamos de población, o cualquier otro tipo de rasgo que divida elementos en grupos de forma discreta y no continua.

A la hora de diseñar el proceso de minería, es importante tener en cuenta por un lado qué tipo de datos se va a utilizar y si existe la necesidad de realizar una traducción de, por ejemplo, valores categóricos a valores numéricos proporcionales y, por otro lado, los distintos rangos en los que se pueden mover los valores numéricos manejados para cada variable. En este último caso, algoritmos de normalización como el *z-scoring* son necesarios en la preparación de los datos a la hora de realizar procesos de minería como es el PCA (*Principal Component Analysis*).

2.1.2. Técnicas de Minería de Datos

Las técnicas de minería de datos son, en la práctica, algoritmos que realizan una determinada función y que ofrecen un resultado. La preparación de los datos de entrada, así como la interpretación de los datos de salida es crucial para determinar si el algoritmo ha realizado la tarea que se necesita. Existen, pues, una amplia gama de algoritmos de minería de datos, y la elección de unos u otros depende tanto de los tipos de datos que se están tratando como de los resultados o el objetivo buscados.

Estas técnicas pueden ser diferenciadas en tres grupos según su funcionalidad: predicción, asociación y agrupamiento o clustering.

La búsqueda de patrones entre los datos puede llevar al descubrimiento de relaciones y correlaciones entre datos que pueden tener carácter asociativo o, a veces, incluso predictivo. El conocimiento predictivo permite aventurar valores futuros de ciertas variables basándose en el pasado.

Por otro lado, la asociación nos permite encontrar los distintos niveles de influencia y correlación entre variables que, a veces, a primera vista no se aprecian.

Por último, las técnicas de clustering o agrupamiento nos permiten encontrar una agrupación de elementos basados en características que pueden ser desconocidas, además, estos algoritmos cuentan con un carácter iterativo y de repetición que los optimizan y buscan la máxima similitud entre elementos de un mismo grupo.

Las principales tecnologías usadas en este proyecto son el Análisis de Componentes Principales (PCA, *Principal Component Analysis*) y el *Clustering*. A continuación se hablará sobre estas dos técnicas.

2.1.2.1 Principal Component Analysis (PCA)

El Análisis de Componentes Principales, ampliamente comentado en [6] y usado en trabajos como [5], [10] y [11], es un algoritmo de reducción de dimensionalidad en un conjunto de datos. Se encarga de detectar, dentro de un gran conjunto de datos con un gran número de atributos, la posible correlación entre éstos y de definir nuevos atributos artificiales que caracterizan de una manera más eficaz a los elementos.

Matemáticamente, el algoritmo escoge un nuevo sistema de coordenadas mediante transformación ortogonal en el cual todas las dimensiones son ortogonales entre sí, representando estas nuevas dimensiones los nuevos atributos del conjunto de datos. Además, la ortogonalidad de las dimensiones implica un grado de correlación nulo entre estos atributos.

Estos nuevos atributos son referidos como Componentes Principales y su característica más importante es que el Primer Componente Principal captura (o explica) el porcentaje de varianza mayor del conjunto de datos original, el Segundo Componente Principal el segundo más grande, y así sucesivamente.

Como resultado, PCA nos devuelve nuestro conjunto de datos original con una nueva dimensionalidad (atributos) entre los cuales no existe correlación y que, partiendo del primero, definen de la mejor manera posible la varianza existente entre los elementos.

En nuestro caso, PCA ayudará a encontrar asociaciones entre los estadísticos de los flujos y generará nuevos atributos (Componentes Principales) que representen de la mejor manera posible la varianza entre ellos, de forma que, al aplicar un algoritmo de *clustering*, sea posible realizar una agrupación de los flujos que represente sus distintos orígenes y ser capaces así de reconocer las distintas aplicaciones.

2.1.2.2 Clustering

Los algoritmos de *clustering* se encargan de encontrar patrones en un conjunto de datos para crear grupos entre ellos basándose en características que incluso pueden ser desconocidas con anterioridad. Se trata de una etapa de gran importancia dentro de todo el proceso de minería de datos, ya que ayuda a agrupar los elementos de manera que el grado de asociación sea máximo dentro de cada clúster y mínimo entre elementos de distintos clusters.

En general, todos los algoritmos de clustering precisan la configuración del número de clusters a buscar y si este número no se conoce de antemano, técnicas heurísticas pueden ser usadas sobre los datos como puede ser el criterio de *Akaike* o el criterio *Bayesiano*.

El algoritmo usado en este proyecto es *K-means* (K-medias), uno de los más usados para realizar agrupamientos debido a su poca complejidad computacional, que ayuda a agrupar grandes cantidades de datos en poco tiempo. El parámetro K del algoritmo indica el número de clusters a buscar y el funcionamiento consiste en elegir K centros de entre todo el conjunto de datos y asignar cada elemento a su centro más cercano según la distancia que se elija (en nuestro caso, utilizaremos la distancia euclidiana).

Los pasos que sigue el algoritmo son los siguientes:

- Paso 1. Dado un número de *centroides* (centros de cluster) K, elegir K puntos de entre todo el conjunto de datos de forma aleatoria.

- Paso 2. Asignar el resto de los puntos al *centroide* más cercano.

- Paso 3. Una vez determinados los K clusters, definir los nuevos centros de dichos clusters en función de los elementos asignados a él.

- Paso 4. Repetir los pasos 2 y 3 de forma reiterada hasta que la recalculación de los nuevos centros pasa a ser estable. Es decir, los centros no se mueven.

El clustering es un proceso muy relevante en este proyecto ya que, una vez realizado el PCA, vamos a comprobar si nuestros elementos se encuentran suficientemente caracterizados por sus atributos como para que, tras realizar el procesado de agrupación, los clusters agrupen en cada uno de ellos los flujos provenientes de cada aplicación.

2.2. Análisis de Tráfico de Internet

2.2.1. Concepto

El análisis, detección y clasificación de tráfico de Internet tiene un gran interés para administradores de redes, empresas y administraciones públicas en las que los recursos son limitados y es conveniente realizar una buena gestión del ancho de banda disponible, así como a limitar qué tipo de aplicaciones son las más adecuadas para usar dicha red según la naturaleza del organismo. Organismos como universidades o medianas o grandes empresas, que necesitan contar con una red no saturada, con una buena gestión del ancho de banda y una rápida capacidad de respuesta, no deberían permitir la posibilidad de que ciertos usuarios abusen de dicho ancho de banda para, por ejemplo, la descarga de contenido multimedia para uso personal.

Desde el inicio de Internet y la proliferación de las redes, ha existido la necesidad de ser capaces gestionar las redes y conocer para qué se usan, sin embargo, al mismo tiempo que las redes se han hecho cada vez más complejas, la complejidad para realizar dichas tareas ha seguido la misma estela. En un principio, organismos como la IANA se encargaron de generar la normativa y la normalización de Internet, incluyendo la asignación puertos UDP y TCP *estándar* para cada aplicación genérica en Internet como, por ejemplo, el puerto 80 para tráfico HTTP, el puerto 443 para HTTPS, puerto 21 para FTP, etc. puertos que reciben el calificativo de “bien conocidos” (*well-known ports*) Esta normalización permite a través de la mera observación de los puertos mediante programas analizadores de tráfico como *Wireshark* identificar y caracterizar de una forma genérica el tráfico.

Hoy en día, buena parte de las aplicaciones usadas a través de Internet son *aplicaciones web* las cuales, normalmente, se comunican mediante HTTPS. Ejemplos de este caso puede ser el *streaming* de video en sitios web como *Youtube*, *streaming* de audio a través de clientes web como *Spotify* o radio, televisión por Internet o servicios de videollamada basados en *VoIP*.

Otras, por ejemplo, usan un modelo de conexión distinto al de cliente-servidor, como pueden ser las redes *Peer-to-Peer* (P2P), en las que cada nodo actúa tanto de cliente como de servidor, además de usar en la mayoría de los casos puertos IP aleatorios o elegidos manualmente por el usuario, los cuales pueden incluso cambiar al inicio de cada ejecución de la aplicación. Este tráfico P2P es actualmente el más usado para intercambio de contenido multimedia o software entre ordenadores, tráfico que, por norma general, mueve grandes cantidades de datos.

Todas estas nuevas aplicaciones hacen de la clasificación de tráfico de Internet una tarea cada vez más compleja y, a su vez, con el exponencial uso de Internet, más necesaria para los administradores de dichas redes.

Esta necesidad ha dado paso al desarrollo de nuevas técnicas dirigidas al análisis de tráfico de Internet. Técnicas como el *Deep Packet Inspection*, *Machine Learning* o técnicas de carácter *heurístico* serán comentadas en la siguiente sección.

2.2.2. Técnicas de Análisis de Tráfico

En los siguientes sub-apartados se describen brevemente algunas de las técnicas para el análisis de tráfico de Internet y que se pueden usar de forma complementaria con la ya comentada observación de puertos y protocolos *well-known*. Muchas de estas técnicas cuentan con un carácter matemático importante, si bien la gran mayoría de ellas son automatizables. En nuestro caso, hemos hecho un uso de técnicas matemáticas de Análisis de Datos (técnicas de preprocesado y normalización, PCA y *clustering*, principalmente), combinadas con algunos métodos pseudo-heurísticos a la hora de determinar qué atributos resultan más adecuados para la identificación y caracterización de los distintos flujos capturados.

2.2.2.1 Deep Packet Inspection

Las técnicas de *Deep Packet Inspection* (DPI), se basan en inspeccionar la parte del paquete incluso más allá de la cabecera, incluida la parte útil de información, *payload*, para recoger información del paquete en busca de virus, spam, intrusiones, criterios para decidir qué hacer con el paquete (en el caso de proveedores de servicios o grandes empresas) y, por último, para la obtención de información estadística. Como se ha comentado anteriormente, es una técnica ampliamente usada por empresas, proveedores de servicios y gobiernos. Se basa en el establecimiento de firmas (*signatures*) creados a partir de dicho contenido, y que permiten la caracterización del tráfico mediante la comprobación para cada paquete o flujo de si cumple o no con las distintas firmas definidas.

2.2.2.2 Machine Learning

Las técnicas de *Machine Learning* son una rama de las ciencias de computación basadas en algoritmos de aprendizaje automático o semiautomático mediante el análisis de parámetros estadísticos característicos de los flujos, haciendo posible predecir o aprender qué tipo de aplicación tiene más probabilidades de haber generado un determinado tráfico en función de dichos parámetros.

2.2.2.3 Técnicas Heurísticas

Las técnicas heurísticas son, como dice una de las definiciones del término “heurístico” según el diccionario de la lengua española de la RAE en su 22ª edición (*manera de buscar la solución de un problema mediante métodos no rigurosos: tanteo, reglas empíricas, etc.*), técnicas de un carácter más informal y personal, basadas en el conocimiento y experiencia propios del propio investigador sobre el tráfico de red en general o incluso, de conocimientos determinados de la red a analizar y qué tipo de aplicaciones y de qué naturaleza pueden estar generando el tráfico.

3. Descripción del proceso de análisis de datos

En primer lugar, se explicará de forma general todo el proceso obtención, preparación y procesado de datos y en las siguientes secciones se analizará cada paso en profundidad.

A modo recordatorio, el objetivo final de este proyecto consiste en ser capaces de identificar las distintas aplicaciones que han generado tráfico dentro de una red, a través de la clasificación de sus flujos.

Partiendo de dicha idea, ha sido necesario contar con capturas de tráfico generadas por aplicaciones conocidas *a priori*, para así poder realizar un proceso de validación de resultados y comprobar el correcto funcionamiento del proceso de clasificación.

Como primer paso, se han creado capturas de tráfico correspondientes a cada aplicación mediante el analizador de tráfico *Wireshark* y el uso de filtros para aislar el tráfico correspondiente a cada aplicación. *Wireshark* nos permite guardar las capturas de tráfico en formato ".pcap", por lo que contaremos con un fichero de captura de tráfico para cada aplicación usada.

El siguiente paso consiste en la agrupación de los paquetes de cada captura en flujos. Un flujo se define como la **combinación** de los siguientes parámetros: *Dirección IP Origen, Dirección IP Destino, Puerto Origen y Puerto Destino* y por último, tráfico *UDP* o *TCP*. Además, se tiene en cuenta un *timeout* (tiempo límite) de duración del flujo a partir del cual se considera un nuevo flujo. Como resultado contaremos con un nuevo archivo ".pcap" para cada flujo en el que se encuentran los paquetes pertenecientes a dicho flujo.

El siguiente paso consiste en la extracción de información de los paquetes pertenecientes a cada flujo. Para obtener dichos parámetros se hace uso de la herramienta *Wireshark* en su versión de comandos (*tshark*). La información extraída es:

- *Tiempo de llegada del paquete (fecha y hora),*
- *Direcciones IP origen y destino,*
- *Puertos UDP o TCP origen y destino,*
- *Longitud (length) del paquete IP en bytes.*

Estos datos son obtenidos para cada uno de los paquetes correspondientes a cada flujo y son guardados en archivos con formato CSV (*comma-separated values*), formato en el que se almacenan los datos en texto plano separados por un carácter a elegir (por defecto el carácter coma ",") obteniendo un nuevo fichero CSV para cada fichero ".pcap". Estos parámetros extraídos de los paquetes de cada flujo se usarán más adelante para calcular estadísticos que junto a PCA nos permitirán la clasificación de los flujos.

Para el cálculo de dichos estadísticos, es necesario agrupar todos esos datos en un mismo lugar para poder procesarlos de forma masiva. Para ello, se ha decidido usar una base de datos MySQL en la que se almacenan los parámetros de cada paquete obtenidos anteriormente.

La carga de información en la base de datos se realiza mediante una clase Java que realiza la lectura iterativa de todos los ficheros CSV y que establece una conexión con la base de datos.

Una vez que contamos con todos los flujos en la base de datos, a partir de los parámetros almacenados para cada paquete se obtienen una serie de estadísticos por cada flujo, que se pueden agrupar en estadísticos de tamaño y estadísticos de tiempo. Además, se distingue entre los dos sentidos en los que los paquetes de cada flujo pueden ir, y que denominaremos de manera simbólica de la siguiente manera: Dirección A y dirección B.

Algunos de los estadísticos de tamaño calculados son: número total de paquetes del flujo, número total de bytes, total de paquetes en dirección A y dirección B, total de bytes en dirección A y dirección B, cálculo de la media y la desviación estándar del número de bytes de cada paquete en cada dirección, etc. Por otro lado, estadísticos de tiempo son: duración total del flujo y cálculo de la media, la desviación estándar y la varianza del tiempo entre paquetes en cada dirección.

Una vez contamos con todos los estadísticos para cada flujo, el siguiente paso consiste en la minería de dichos datos para la que utilizaremos la herramienta RapidMiner.

En primer lugar se realiza una preparación de los datos entre los que se encuentran procesos de normalización de los valores o la detección de *outliers* (elementos lejanos dentro del conjunto de datos) que pueden introducir ruido en los procesos siguientes. A continuación, se realizan procesos propios de la minería de datos como el Análisis de Componentes Principales (*PCA*) y el *clustering* (agrupamiento) de los elementos. Este último paso de agrupación es el que nos permite clasificar los datos en distintos grupos (*clusters*) que nos permitirán asignar cada flujo a una aplicación determinada.

El último paso consiste en la validación de los resultados obtenidos. Para ello se hará uso de la información de la que se dispone en la base de datos en la que contamos con la clasificación real de cada flujo con su aplicación y que podremos comparar con los resultados obtenidos en el *clustering* de los datos.

Para ilustrar el grado de acierto de los resultados obtenidos, se confecciona una Matriz de Confusión (*confusion matrix*) a través de la cual se calculan parámetros como son la *precisión*, el *recall*, el *accuracy*, o el *F-measure*, parámetros que nos ayudan a entender y cuantificar la similitud de los resultados del análisis de datos con respecto a la verdadera pertenencia a las distintas aplicaciones.

La figura 1 ilustra un diagrama de flujo representando todo el proceso descrito anteriormente.

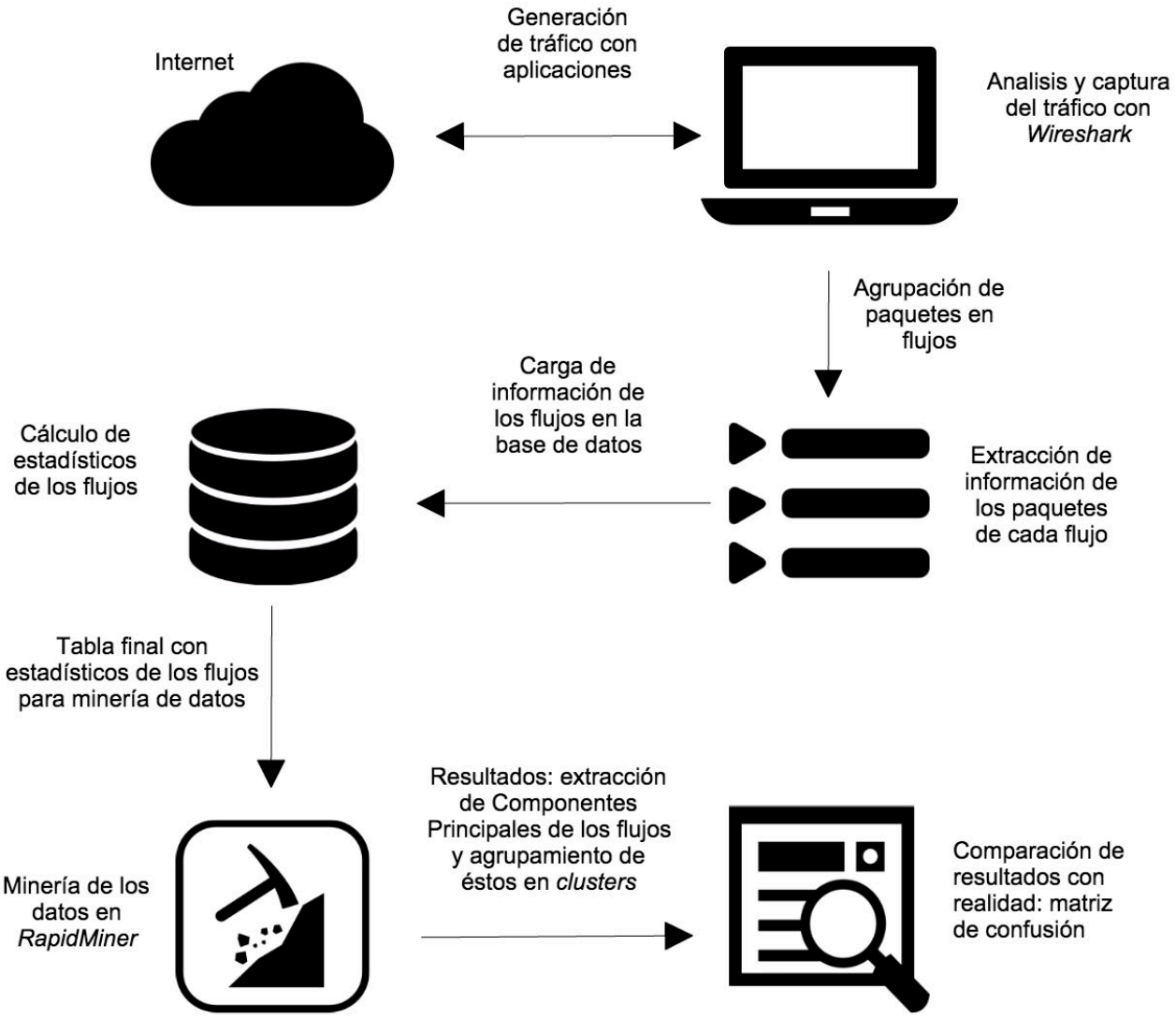

Figura 1. Diagrama de flujo del procesado de datos

3.1. Herramientas software para minería de datos

Existe una gran variedad de herramientas software capaces de realizar funciones de Minería de Datos, muchas de ellas *open-source*. Ejemplos de estas aplicaciones pueden ser *RapidMiner* [12] (que explicaremos con más detalle ya que es la utilizada en este proyecto), *Orange* [13], *Weka* [14] o *KNIME* [15].

RapidMiner es una de las más potentes de todas y se trata de una plataforma software que cuenta con un entorno de aprendizaje integrado, además de realizar funciones como minería de datos, minería de texto, análisis de negocio y análisis predictivo, puede ser usada tanto para usos industriales como para fines académicos. Cuenta con una gran comunidad y un gran soporte y está programada en Java, contando con un gran número de algoritmos disponibles que son representados como módulos o bloques y siendo posible la creación de nuevos módulos a través del entorno de desarrollo que ellos mismos proporcionan.

Los *procesos* de *RapidMiner* están formados por bloques o módulos, de los que cuenta con más de 500, siendo cada uno de estos un paso del proceso y estando todos ellos enlazados mediante conexiones. Estos módulos pueden implementar desde un algoritmo de minería de datos, hasta una operación de reemplazado, de escritura o lectura de datos, entre otros. Además, cada bloque sólo precia la configuración de ciertos parámetros que dependen del algoritmo que implementan, lo que permite la utilización del programa sin la necesidad de escribir código. En la figura 2 podemos ver un ejemplo de proceso en *RapidMiner* que contiene tres módulos: lectura de datos de un fichero CSV, normalización de los datos y *clustering*.

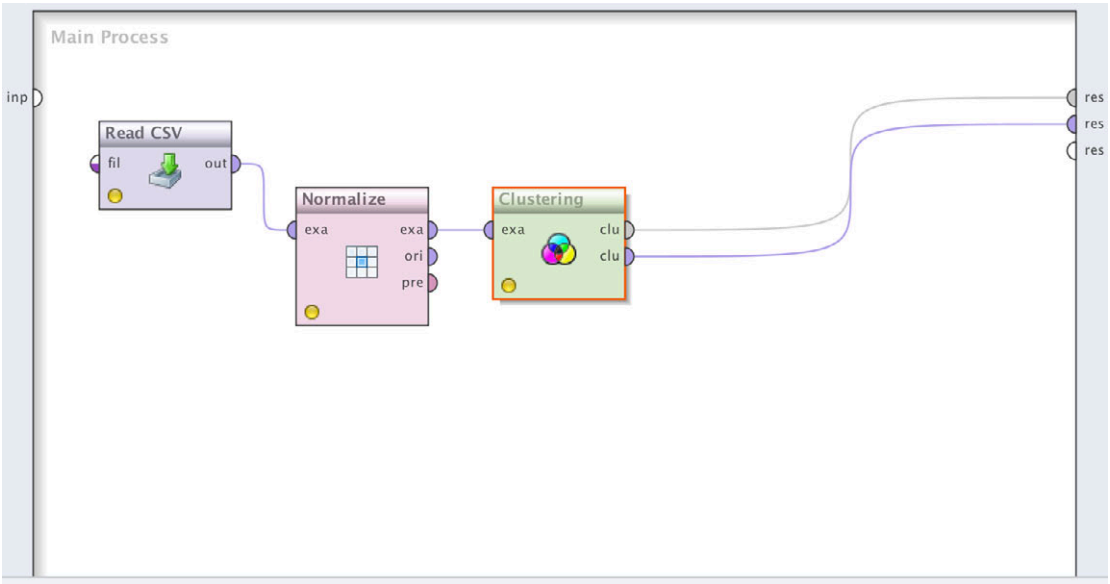

Figura 2. Ejemplo de proceso en RapidMiner

Además, permite la entrada de datos en formato CSV, Excel y TAB, y soporta conexiones con la mayoría de las tecnologías fundamentales de bases de datos SQL como Oracle, MySQL, Microsoft SQL Server, Postgre SQL, etc, y siendo esta conexión totalmente transparente al usuario, por lo que no se precisan conocimientos SQL para su uso.

Hasta la versión 5.3, *RapidMiner* era un proyecto *open-source* alojado en *SourceForge* hasta que, a partir de la versión 6.0, se transformó en producto comercial explotado por la empresa Rapid-I. Actualmente, en su versión 6.3, cuenta con varios formatos. Desde la versión gratuita *RapidMiner Studio*, la cual contiene límites como, por ejemplo, el uso de 1GB de memoria RAM, hasta la versión *Enterprise*, pasando por la versión *Professional* que, a la fecha de escritura de este documento, cuenta con un precio de suscripción mínimo de 2.999$ anuales.

Por otro lado, *RapidMiner* cuenta con la posibilidad de ser conectada con otras herramientas como *WEKA* y poder acceder a los diferentes algoritmos que éstas poseen, haciendo a *RapidMiner* una herramienta de gran flexibilidad.

RapidMiner es la herramienta elegida para realizar este proyecto. Debido a las limitaciones de las versiones gratuitas de la versión 6.0 en adelante, se usará la versión 5.3, que cuenta con todos los módulos y algoritmos necesarios para nuestro proyecto, además de la potencia suficiente para el volumen de datos con el que trabajaremos.

3.2. Descripción del entorno usado

La realización de este proyecto se ha realizado bajo el sistema operativo Mac OS X, sistema basado en Linux compatible con todas las herramientas usadas ya que la mayoría de ellas son multiplataforma.

En primer lugar, se ha hecho uso de la herramienta de captura de tráfico *Wireshark* [16] tanto en su versión de escritorio, como en su versión en línea de comandos *tshark* [9] para realizar las capturas de tráfico sobre las que se han hecho las pruebas, además de para la extracción de parámetros de los paquetes de cada flujo. La versión usada de *Wireshark* ha sido 1.12.5, versión disponible tanto en Windows como en OS X.

También se ha hecho uso del lenguaje de programación Java para escribir un pequeño programa que realiza la carga de información en la base de datos, usando el entorno de desarrollo *Java Platform JDK Version 8 Update 45*.

En cuanto a la base de datos usada, se ha elegido MySQL en su versión abierta y gratuita *Community Edition* [17], en la que, además de haber usado el servidor, se ha hecho uso del software *MySQL Workbench* para gestionar la base de datos, además de para realizar la ejecución de los scripts SQL. La versión de servidor MySQL ha sido la *5.6.23 MySQL Community Server (GPL),* y la versión del *Workbench*, la 6.2. Ambos programas se encuentran también disponibles para la plataforma Windows.

También se ha hecho uso del interprete de comandos Bash para la creación de varios scripts con el fin de automatizar ciertas tareas. Bash es un programa ampliamente extendido que cuenta con versiones para todas las plataformas.

Otro programa importante para la realización del proyecto ha sido *pkt2flow*, un pequeño software creado por Xiaming Chen [7]. Este software es usado para dividir archivos de captura de tráfico de *Wireshark* en flujos de forma automática. Es compatible con cualquier sistema Linux.

Por último, la versión 5.3.013 de *RapidMiner* [12] ha sido la elegida para realizar las tareas de minería de datos. En dicha versión *RapidMiner* aún era software libre y aunque hoy en día está algo anticuado, es suficientemente potente y estable para realizar las tareas necesarias, además de encontrarse sin las limitaciones propias de las versiones de evaluación. *RapidMiner* es ejecutado bajo la plataforma Java, por lo que puede ser ejecutado en cualquier plataforma. Para la ejecución de *RapidMiner* se ha escrito un pequeño script en Bash con un comando Java que permite su ejecución junto con la un parámetro que permite al programa el uso de 3 GB de memoria RAM, ya que, por defecto, se ejecuta con 1 GB y, en algunas ocasiones, esta cantidad no es suficiente. El script se muestra en el Cuadro 1.

```
#!/bin/bash
java -Xmx3000m –jar rapidminer/lib/rapidminer.jar
```

Cuadro 1. Script para la ejecución de *RapidMiner*

Para reproducir todo el proceso desarrollado en este proyecto, se recomienda al lector el uso de una máquina virtual de un sistema operativo basado en Linux, como por ejemplo Ubuntu, plataforma bajo la que todas las herramientas software usadas funcionan a la perfección.

Nota importante: nótese que, en las siguientes secciones se muestran scripts y programas en diferentes lenguajes en los que en ocasiones se usan rutas del sistema de archivos a carpetas y archivos locales propias del ordenador en el que se realizó la ejecución de dichos scripts. Si el lector desea replicar dicha ejecución, será necesario reemplazar las rutas por las propias de su sistema de archivos.

3.3. Preparación de las capturas de tráfico

En un entorno real no sería necesario preparar las capturas de tráfico ya que se partiría de una ya obtenida procedente de la red de la cual se quiere analizar el tráfico. Sin embargo, en el desarrollo de este proyecto ha sido necesario generar el tráfico de cada aplicación de forma separada e intencionada con el fin de ser capaces de asociar cada flujo a una determinada aplicación y poder comprobar posteriormente la confiabilidad de la clasificación obtenida.

Hemos seleccionado dos aplicaciones para realizar los análisis, que representan de manera adecuada por un lado tráfico muy habitual y generalmente permitido en entornos empresariales y académicos (*Tráfico Web*) y por otro lado tráfico *peer-to-peer* utilizado para el intercambio de ficheros entre particulares, en muchas ocasiones para uso personal (*Tráfico Torrent P2P*). Se trata de una primera selección que permitirá verificar la aplicabilidad de las técnicas seleccionadas a estos dos conjuntos de aplicaciones, y que en trabajos futuros se podría ampliar a un repertorio mayor. Para la preparación de las capturas de tráfico se ha hecho uso de *Wireshark* para capturar el tráfico generado por un navegador web, y el tráfico generado por un cliente torrent.

3.4. Preprocesado y filtrado en Wireshark

Para capturar el tráfico de las aplicaciones, se ha usado *Wireshark*. Para cada aplicación se ha creado un fichero de captura independiente que contiene exclusivamente tráfico perteneciente a la aplicación correspondiente. Para ello, se han usado filtros de *Wireshark* para aislar dicho tráfico.

Por ejemplo, para filtrar el tráfico web, se ha usado el siguiente filtro:

tcp.port == 443 || tcp.port == 80

Este filtro configura la captura de todo el tráfico que usa los puertos TCP 443 o 80, siendo éstos los puertos correspondientes a HTTPS y HTTP, respectivamente. Para capturar el tráfico *torrent*, por otro lado, se han definido los filtros análogos según el puerto que el cliente *torrent* usaba en ese momento.

Otras consideraciones a tener en cuenta sobre las capturas realizadas es que han sido considerablemente extensas, de más de 200.000 paquetes, y con una duración mínima de 20 minutos. En la figura 3 se puede observar un ejemplo de capturas de tráfico *Web* y *Torrent*.

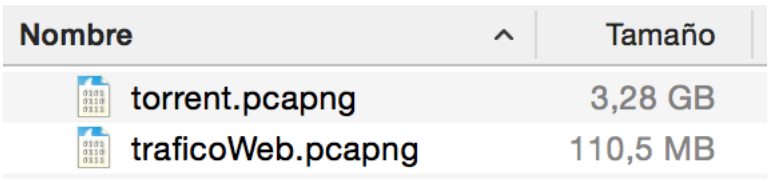

Figura 3. Ejemplo de capturas de dos aplicaciones

3.5. División en flujos y extracción de parámetros

Para la división en flujos se ha hecho uso de [7], se trata de un pequeño software muy ágil llamado "*pkt2flow*" para la división de capturas de tráfico en formato ".pcap" en flujos. La funcionalidad consiste en que, dado un fichero de captura y tras configurar ciertos parámetros como el *timeout* de los flujos (tras este *timeout* se consideraría un flujo nuevo), el programa es capaz de realizar una división de flujos distinguiendo entre tráfico TCP y UDP, agrupando los flujos mediante la combinación única de los siguientes parámetros: *Dirección IP Origen, Dirección IP Destino, Puerto Origen y Puerto Destino.*

Al tratarse de una operación en bucle en la que es necesario realizar una ejecución en comando para cada una de las capturas, se ha escrito un pequeño script en *bash* para realizar dicha tarea en el que se ejecuta el programa *pkt2flow* de forma iterativa para cada captura de tráfico (ver Cuadro 2).

```
#!/bin/bash
FILES=/Users/luisgil/Desktop/PFG/capturas/*.pcapng
for f in $FILES
do
  filename="${f##*/}"
  echo "Processing $f file in $filename"
  /Users/luisgil/Desktop/PFG/pkt2flow/pkt2flow  -u  -o
/Users/luisgil/Desktop/PFG/divisiones/$filename $f
done
```

Cuadro 2. Script bash para la división automática en flujos

Como se puede observar, el script realiza la tarea para cada fichero con extensión .pcapng que se encuentre dentro de la carpeta "capturas" (se trata de una sub-carpeta de "/Users/luisgil/Desktop/PFG/"; ver la nota importante incluida al final de la sección 3.2 de esta memoria) y ejecuta el programa "*pkt2flow*" determinando la carpeta de salida a través del parámetro "-o" como *divisiones/nombredearchivo*. Además, se incluye el parámetro "-u" para indicar al programa que deseamos que incluya también los flujos UDP. Por último, es importante detallar que en el programa *timeout* de los flujos ha sido configurado en 10 minutos.

En la figura 4 podemos ver el resultado de la ejecución de dicho script en lo que respecta a los ficheros generados por el mismo.

En la figura 4 podemos ver el resultado de la ejecución de dicho script.

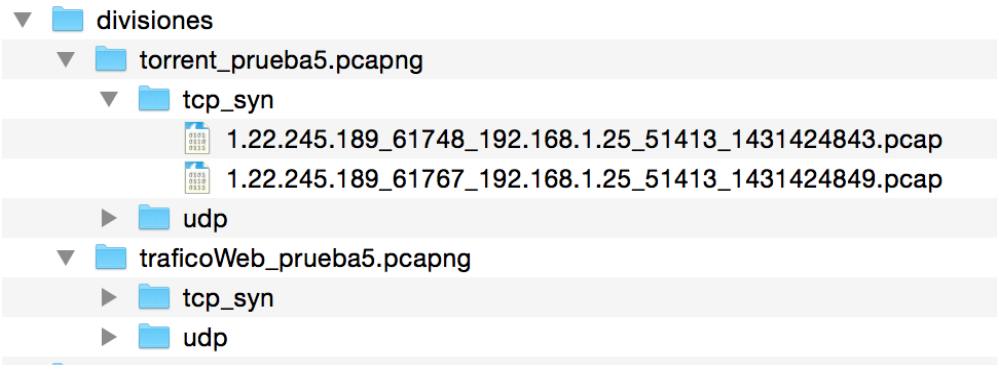

Figura 4. Ejemplo de resultado de ejecución del programa "pkt2flow"

Una vez contamos con la división en flujos de cada una de las capturas, el siguiente paso es extraer para cada paquete de cada flujo parámetros característicos como son el tiempo de llegada del paquete (fecha y hora), la dirección IP origen y destino, puertos UDP y TCP tanto origen como destino y la longitud (*length*) del paquete IP. Para realizar esta tarea se hace uso de la herramienta *tshark* que nos proporciona *Wireshark*, lo cual no es más que la versión en línea de comandos del programa. En [9] podemos encontrar la documentación de dicha herramienta y en este caso se hace uso de la opción "*-T fields*" que nos permite, mediante el comando "*-e field*" introducir cualquiera de los campos pertenecientes a cualquiera de los protocolos de red soportados por *Wireshark*, los cuales pueden ser encontrados en [8].

En nuestro caso, el comando a utilizar sobre cada uno de los flujos será el que se muestra en el Cuadro 3, el cual nos dará una salida con el formato ejemplificado en el Cuadro 4.

```
tshark -r captura.pcapng -T fields -e frame.time_epoch -e ip.src
-e tcp.srcport -e udp.srcport -e ip.dst –e tcp.dstport -e
udp.dstport -e ip.len -E header=n -E separator=%
```

Cuadro 3. Comando para extraer parámetros de los flujos

```
1431423173.633212000%192.168.1.25%50364%%17.167.194.123%443%%64
1431423174.654497000%17.167.194.123%443%%192.168.1.25%50364%%52
1431423174.654590000%192.168.1.25%50364%%17.167.194.123%443%%40
```

Cuadro 4. Ejemplo de salida de extracción de parámetros de un flujo

Donde cada línea representa los datos de cada paquete perteneciente al flujo, estando estos separados por el carácter "%".

Además, para realizar la tarea de forma automática, y dada la gran cantidad de flujos presentes en cada captura, se ha escrito otro script de estructura similar al anterior para realizar esta tarea (ver Cuadro 5). Además, cada resultado es guardado en un fichero CSV con el mismo nombre que su correspondiente fichero .pcapng.

```
#!/bin/bash
FILES=/Users/luisgil/Desktop/PFG/divisiones/*/*/*.pcap
for f in $FILES
do
  echo "Processing $f file..."
  tshark -r $f -T fields -e frame.time_epoch -e ip.src -e
tcp.srcport -e udp.srcport -e ip.dst -e tcp.dstport -e
udp.dstport -e ip.len -E header=n -E separator=% > $f.csv
done
```

Cuadro 5. Script bash para la extracción de parámetros de los flujos

Un ejemplo del resultado de la ejecución del script anterior se puede ver en la figura 5. Para cada flujo, se crea un archivo CSV que contiene la información deseada de cada flujo.

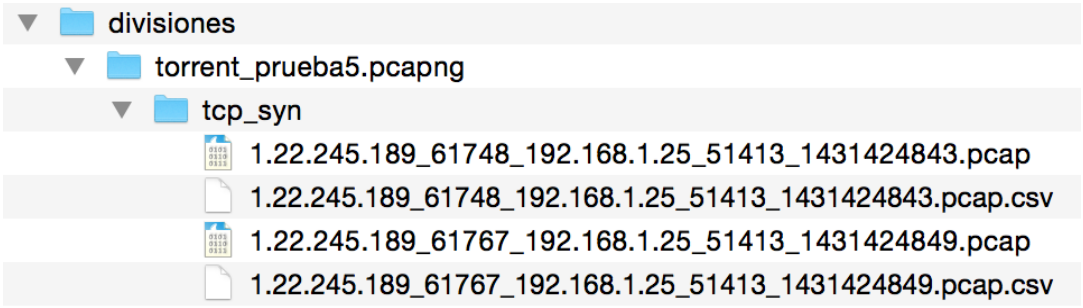

Figura 5. Ejemplo de resultado de la extracción de parámetros de los flujos

3.6. Carga y procesado de datos en base de datos MySQL

El siguiente paso a realizar una vez hemos obtenido los datos necesarios de cada flujo, es realizar la carga de dichos datos a la base de datos creada para tal efecto. Para ello, primero es necesario definir dicha base de datos que contará con 3 tablas principales: *Aplications*, *Clasification* y *Packets*. En la primera tabla *Aplications* se incluirá el nombre de cada una de las capturas originales procedentes de cada aplicación junto con su *AplicationID*. En la segunda tabla, *Clasification*, se cruzarán cada ID de cada flujo *FlowID*, con el *AplicationID* (es decir, la aplicación) a la que pertenece dicho flujo. Por último, en la tabla *Packets*, se insertarán todos y cada uno de los paquetes de cada uno de los flujos, indicando para cada paquete, entre otras cosas, el *FlowID* que les corresponde. El siguiente script SQL (Cuadro 6) muestra la definición de las 3 tablas principales.

```
CREATE TABLE `Packets` (
  `FrameNumber` int(11) NOT NULL AUTO_INCREMENT,
  `FrameTimeEpoch` varchar(45) NOT NULL,
  `IPSrc` varchar(45) NOT NULL,
  `TCPSrcport` varchar(45) DEFAULT NULL,
  `UDPSrcport` varchar(45) DEFAULT NULL,
  `IPDst` varchar(45) NOT NULL,
  `TCPDstport` varchar(45) DEFAULT NULL,
  `UDPDstport` varchar(45) DEFAULT NULL,
  `IPLength` varchar(45) NOT NULL,
  `FlowID` int(11) NOT NULL,
  `FileName` varchar(200) NOT NULL,
  PRIMARY KEY (`FrameNumber`)
) ENGINE=InnoDB AUTO_INCREMENT=84820 DEFAULT CHARSET=latin1;

CREATE TABLE `Clasification` (
  `FlowID` int(11) NOT NULL,
  `AplicationID` int(11) NOT NULL,
  PRIMARY KEY (`FlowID`,`AplicationID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `Aplications` (
  `AplicationID` int(11) NOT NULL AUTO_INCREMENT,
  `AplicationName` varchar(45) NOT NULL,
  PRIMARY KEY (`AplicationID`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;
```

Cuadro 6. Script SQL para la creación de las tablas iniciales de la base de datos

Para realizar la carga de todos los paquetes en la base de datos, se usarán los archivos CSV creados anteriormente para cada uno de los flujos, y se realizará mediante la ejecución de métodos de la clase "*CargaBaseDatos*" escrita en Java. El código de dicha clase puede ser encontrado en el Anexo A.

Este programa se encarga de analizar la carpeta "*divisiones*" (cuya estructura podemos ver en la figura 4) y establecer una conexión con la base de datos para después analizar todos los archivos dentro de las subcarpetas y procesar aquellos archivos que tengan formato CSV. En concreto, el programa realiza estas tres funciones principales:

- 1. Para cada captura ".pcap" original, añade una nueva entrada en la tabla *Aplications* con el nombre de dicha captura y le asigna un *AplicationID*. En la figura 6 podemos ver un ejemplo de la tabla *Aplications*.

- 2. Para cada nuevo flujo, se le asigna su correspondiente *AplicationID* dentro de la tabla *Clasification*. En la figura 7 podemos ver un ejemplo de la tabla *Clasification*.

- 3. Para cada archivo ".pcap" correspondiente a un flujo, se añaden todos los datos de los paquetes pertenecientes a dicho flujo en la tabla *Packets*, junto con el *FlowID* que identifica al flujo al que pertenecen. En la figura 8 podemos ver un ejemplo de la tabla *Packets*.

	AplicationID	AplicationName
▶	1	trafico_Torrent
	6	trafico_Web

Figura 6. Tabla *Aplications* tras la ejecución de los métodos de la clase Java *CargaBaseDatos*

	FlowID	AplicationID
▶	1	1
	2	1
	3	1
	4	1
	5	1
	6	1
	7	1
	8	1
	9	1

Figura 7. Tabla *Clasification* tras la ejecución de los métodos de la clase Java *CargaBaseDatos*

FrameNumber	FrameTimeEpoch	IPSrc	TCPSrcport	UDPSrcport	IPDst	TCPDstport	UDPDstport	IPLength	FlowID
1	1431425577.194491000	151.45.165.17	50206		192.168.1.25	51413		40	1
2	1431425577.194554000	192.168.1.25	51413		151.45.165.17	50206		552	1
3	1431425577.194555000	192.168.1.25	51413		151.45.165.17	50206		552	1
4	1431425577.194555000	192.168.1.25	51413		151.45.165.17	50206		552	1

Figura 8. Tabla *Packets* tras la ejecución de la clase Java *CargaBaseDatos*

Tras finalizar la ejecución del programa contamos con todos los paquetes clasificados por flujos y por aplicaciones dentro de nuestra base de datos. El objetivo de procesar los datos dentro de la base de datos es, como ya se ha comentado, obtener una tabla final con parámetros estadísticos que caractericen cada flujo de cada aplicación. Los parámetros que se obtienen son los siguientes:

- **Estadísticos identificativos:**
  - ***FlowID***: ID única que identifica al flujo.
- **Estadísticos de tiempo:**
  - ***FlowDuration***: Duración del flujo en segundos.
  - ***TimesAverageDirectionA***: Cálculo de la media de tiempos entre un paquete y el paquete anterior en la dirección A.
  - ***TimesStandardDeviationDirectionA***: Cálculo de la desviación estándar de tiempos entre un paquete y el paquete anterior en la dirección A.
  - ***TimesVarianceDirectionA***: Cálculo de la varianza de tiempos entre un paquete y el paquete anterior en la dirección A.
  - ***TimesAverageDirectionB***: Cálculo de la media de tiempos entre un paquete y el paquete anterior en la dirección B.
  - ***TimesStandardDeviationDirectionB***: Cálculo de la desviación estándar de tiempos entre un paquete y el paquete anterior en la dirección B.
  - ***TimesVarianceDirectionB***: Cálculo de la varianza de tiempos entre un paquete y el paquete anterior en la dirección B.

- **Estadísticos de tamaño:**

  - ***TotalFlowPackets***: Número total de paquetes en el flujo.
  - ***TotalFlowBytes***: Número total de bytes transmitidos en el flujo, en ambas direcciones.
  - ***PacketsDirectionA***: Número de paquetes enviados en la dirección A.
  - ***TotalBytesDirectionA***: Número total de bytes enviados en la dirección A.
  - ***AverageBytesDirectionA***: Cálculo de la media del tamaño de los paquetes enviados en la dirección A.
  - ***BytesStandardDeviationDirectionA***: Cálculo de la desviación estándar del tamaño de los paquetes enviados en la dirección A.
  - ***BytesVarianceDirectionA***: Cálculo de la varianza del tamaño de los paquetes enviados en la dirección A.
  - ***PacketsDirectionB***: Número de paquetes enviados en la dirección B.
  - ***TotalBytesDirectionB***: Número total de bytes enviados en la dirección B.
  - ***AverageBytesDirectionA***: Cálculo de la media del tamaño de los paquetes enviados en la dirección B.
  - ***BytesStandardDeviationDirectionB***: Cálculo de la desviación estándar del tamaño de los paquetes enviados en la dirección B.
  - ***BytesVarianceDirectionB***: Cálculo de la varianza del tamaño de los paquetes enviados en la dirección B.
  - ***PacketsRelationAtoB***: Relación entre el número de paquetes enviados en la dirección A y la dirección B.
  - ***BytesRelationAtoB***: Relación entre el número total de bytes enviados en la dirección A y la dirección B.
  - ***BytesPerPacketDirectionA***: Relación entre el número de paquetes y el número de bytes enviados en la dirección A.
  - ***BytesPerPacketDirectionB***: Relación entre el número de paquetes y el número de bytes enviados en la dirección B.

Para conseguir tal fin, se ha desarrollado un script en lenguaje SQL que usa como base las tablas iniciales ya creadas, y que ya contienen los datos de los flujos, *Aplications*, *Clasification* y *Packets*, para crear toda una serie de tablas intermedias hasta obtener la llamada *FinalTable* que será la que introduzcamos en *RapidMiner*. El script SQL puede ser encontrado en el Anexo B de este documento.

Del script SQL que podemos encontrar en el Anexo B cabe destacar la variable "*minimum_Packets*". Este valor indica el número mínimo de paquetes que debe tener un flujo para ser considerado válido. Si un flujo esta formado por menos del número de paquetes especificado, es descartado y no aparecerá en la *FinalTable*.

Durante el script, se crean numerosas "vistas SQL" (*SQL View*) que dan como resultado tablas auxiliares con diversos parámetros que son usados para calcular la tabla final. Uno de los parámetros más importantes que se calculan es la dirección del paquete entre todos los paquetes que puede formar un flujo. Para ello, se definen dos direcciones: *Direction A* y *Direction B*, y, para cada flujo, se considera la *Direction A* aquella que contenga mayor cantidad de bytes de información. Este método de división en direcciones ha sido considerado el más óptimo considerando el uso final del proyecto, que es analizar trazas de tráfico de nodos de Internet con gran cantidad de paquetes. Se considera que en una estructura cliente-servidor, suponer que la dirección en la que se observa más volumen de datos es la que corresponde al flujo con origen en el servidor y destino en el cliente dará unos resultados en muchas ocasiones acordes con la realidad, aunque esto depende de la aplicación concreta.

Como último paso, la tabla *FinalTable* resultado es exportada a un archivo en formato CSV mediante la opción que nos brinda *MySQL Workbench*. Dicho archivo CSV será importado en el siguiente paso en el proceso de *RapidMiner* para posteriores análisis.

3.7. Minería de los datos en RapidMiner

Se ha diseñado un proceso en RapidMiner para la minería de los datos que cuenta con diferentes módulos. Los procesos de RapidMiner son definidos en XML para ofrecer una fácil portabilidad además de edición. En el Anexo C de este documento se puede encontrar el XML correspondiente al proceso diseñado para este proyecto. La figura 9 ilustra los bloques que son usados durante el proceso.

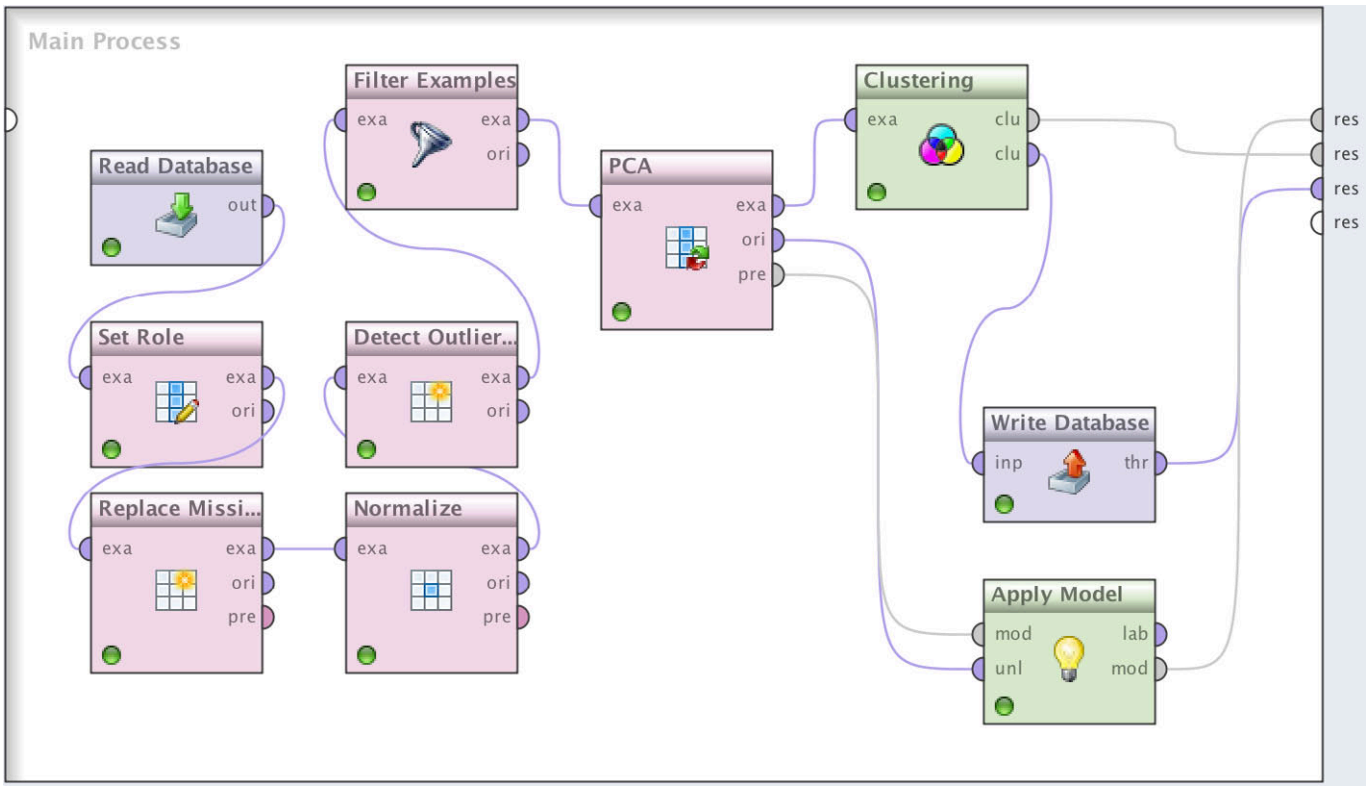

Figura 9. Proceso de minería de los datos en *RapidMiner*

Los pasos que se realizan durante este proceso son:

- *Lectura de datos en la base de datos (Read Database) y cambio del tipo de parámetro del identificador de flujo (Set Role)*
- *Reemplazado de los valores NULL (Replace Missing Values)*
- *Normalización de los datos mediante la técnica z-scoring (Normalize)*
- *Detección de outliers (elementos lejanos del conjunto de datos) (Detect Outliers)*
- *Filtrado de outliers (Filter Examples)*
- *PCA*
- *Clustering*
- *Escritura de resultados en la base de datos (Write Database)*

Cada módulo del proceso cuenta con varios parámetros de configuración que son relevantes en el proceso. A continuación se ampliará la información sobre cada uno de ellos.

- **Módulo *Read Database***

Se encarga de obtener los datos que van a ser usados durante todo el proceso y que deben tener formato de tabla. Se realiza una conexión con la base de datos y se realiza la siguiente *Query SQL*:

*SELECT * FROM FinalTable*

Donde, recordemos, *FinalTable* es la tabla donde se encuentran los estadísticos de los flujos.

- **Módulo *Set Role***

La función de este modulo es muy simple: cambia el *rol* del atributo *FlowID* de la tabla *FinalTable* a "*label*" (etiqueta) simplemente para evitar que el resto del proceso tenga en cuenta dicho valor y no lo use como valor numérico, ya que sólo sirve para identificar al flujo.

- **Módulo *Replace Missing Values***

Este módulo es el más simple de todos. Su única función es la de reemplazar los valores de la tabla de datos que estén a *NULL* con el valor 0. Los campos en los que pueden existir valores a *NULL* en la tabla de datos son aquellos correspondientes a algunos parámetros estadísticos relativos a la *Direction B*, y esto ocurrirá en aquellos flujos que sólo tengan paquetes en la *Direction A*, escenario que, aunque sea de escasa ocurrencia, es necesario contemplar.

- **Módulo *Normalize***

Los datos presentes en la tabla de datos pueden ser muy variados, y mientras que hay flujos con, por ejemplo, miles de paquetes, otros flujos tienen del orden de 40 o 50. Esto hace necesaria una normalización de los datos de forma que los rangos entre máximos y mínimos se haga lo más pequeña posible, práctica que, como vemos en [6], es muy importante a la hora de realizar el PCA: una buena preparación de los datos puede cambiar por completo los resultados.

El algoritmo utilizado en este módulo es el *z-scoring*, también llamado "normalización estadística", y que consigue adaptar el conjunto de datos a los parámetros de una distribución Normal de media = 0 y varianza = 1. Funciona de la siguiente forma: calculando por columnas (es decir, por atributos), cada nuevo valor es calculado siguiendo la fórmula Z = (X-u)/s, siendo Z el nuevo valor, X el valor original, u la media de todos los valores de la columna correspondiente y s la desviación estándar. Esta normalización implica que, en un rango de -3 a +3 veces la desviación estándar del atributo en cuestión, el 99.9% de los datos está representado.

- **Módulo *Detect Outliers* y *Filter Examples***

Se define como *outliers* aquellos elementos del conjunto de datos que, medidos mediante distancia euclidiana, se encuentran significativamente lejos del resto, es decir, son "lejanos a la mayoría". Estos elementos pueden representar casos extremos o incluso erróneos que sin embargo podrían distorsionar el resultado final del análisis de datos, por lo que se suelen eliminar. El módulo *Detect Outliers* se encarga de detectar estos elementos y cuenta con dos parámetros a configurar:

  - ***Number of neighbors***: número de los vecinos más cercanos con los que cada punto es comparado (calculada su distancia). En nuestro caso el número elegido es 3, la cual es la configuración recomendada ya que, ante un número mayor, la computación necesaria aumenta exponencialmente y además 3 vecinos se consideran suficiente para determinar si un elemento es lejano.

  - ***Number of outliers***: número total de *outliers* a buscar. Este valor depende del número total de elementos con los que se trabaje. En las pruebas realizadas se ha buscado un número de *outliers* correspondiente al 1% del total de elementos dentro del conjunto de datos. Dependiendo del número de flujos con el que se trate, el número de *outliers* a buscar debe ser introducido en la configuración.

El módulo *Detect Outliers* etiqueta aquellos elementos como tal dando un valor a la etiqueta "*outlier*" igual a "*true*". Luego, en el módulo *Filter Examples*, se conservan sólo aquellos elementos que tengan la etiqueta "*outlier*" con el valor "*false*". En la figura 10 se pueden ver los parámetros configurados en el módulo *Detect Outliers* de *RapidMiner* en el caso de nuestros experimentos.

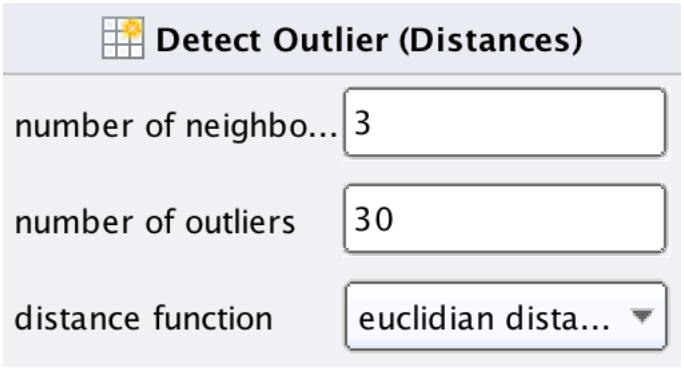

Figura 10. Parámetros configurados en el módulo *Detect Outliers*

- **Módulo PCA (*Principal Component Analysis*)**

Se trata uno de los dos módulos principales del proceso de minería, junto con el de *Clustering*. Se ocupa de realizar el análisis de componentes principales en sí y, como resultado, entrega una nueva tabla de datos dimensionada en función de los nuevos componentes principales que reciben el nombre pca_1, pca_2, etc.

El módulo PCA cuenta con un sólo parámetro a configurar relativo a la reducción de dimensiones, llamado "*dimensionality reduction*". En este caso se ha elegido el valor "*keep_variance*" y junto con ésta opción es necesario indicar el "*variance threshold*", el cual esta configurado con un valor de 0.95.

Esta configuración implica que el módulo PCA sólo incluirá en la tabla resultado aquellos componentes que de manera conjunta acumulen el 95% de la varianza de los datos. Una vez se haya definido un número de componentes que acumulen el 95% de la varianza, el resto será descartado. En otras palabras, este parámetro determina el número de componentes principales que PCA encontrará. En la figura 11 se puede observar una captura de la configuración del módulo.

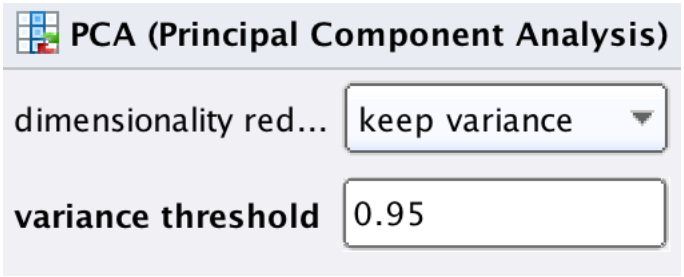

Figura 11. Parámetros configurados en el módulo *PCA*

- **Módulo *Clustering***

El *clustering* es el último y uno de los más importantes procesos que se realizan y consiste en intentar identificar agrupaciones (*clusters*) de elementos entre el set de datos. Para ello, primero es necesario especificar el parámetro "*k*" que identifica el número de centroides, y por tanto de clusters, a identificar. En nuestro caso, puesto que buscamos distinguir entre dos aplicaciones, debemos seleccionar 2.

El algoritmo usado es *k-means*, uno de los algoritmos más habituales para este cometido, debido a su simplicidad y a su rápida ejecución. Se basa en un proceso reiterativo en el que se eligen de forma arbitraria k centroides y el resto de elementos son asignados a uno de ellos según la distancia euclidiana hasta formar los clusters. Luego, nuevos centros basados en los clusters formados son calculados y se vuelve a realizar el proceso hasta que al final, los centros no se mueven. La figura 12 muestra los parámetros elegidos para la configuración del módulo de *clustering* para el caso de k=2.

Figura 12. Parámetros configurados en el módulo *Clustering*

- **Módulo *Write Database***

Este último modulo se encarga de escribir en la base de datos los resultados del proceso. Para ello, se crea una nueva tabla en la base de datos llamada "*ResultadosClustering*".

Como resultado del proceso de minería de los datos, obtenemos una nueva tabla en la que a cada flujo se le ha asignado una etiqueta indicando el clúster en el que ha sido clasificado. Por último, queda comparar los resultados con los que realmente conocemos acerca de la aplicación correspondiente a cada flujo, y para ello usaremos una matriz de confusión.

- **Módulo *Apply Model***

Se trata de un módulo auxiliar usado para interpretar los datos arrojados por el módulo PCA. Como se puede observar, el módulo PCA cuenta con varias salidas, una de las cuales es un "modelo *RapidMiner*" que debe ser introducido en este tipo de módulo para ser observado. En la practica, este módulo no tienen ninguna función activa.

3.8. Comparación de resultados: matriz de confusión

El método elegido para la comparación de resultados ha sido la Matriz de Confusión ya que consigue expresar de manera muy visual y resumida la importancia tanto de los aciertos como de los errores en la clasificación de los flujos. Para ello, se han calculado parámetros como el *Recall*, la *Precision*, o el *F-measure*, además del *Accuracy* global. A continuación se extiende un poco más el significado de estos parámetros. En la figura 13 podemos encontrar un ejemplo de matriz de confusión en el que se comparan los resultados de clasificación entre tráfico *torrent* y tráfico web y la realidad.

- ***Accuracy***: medida global de la precisión de la clasificación, basado en el total de elementos.

- ***Precision***: se calcula para cada aplicación y se denomina "valor positivo predicho". Se define como "la fracción de los elementos identificados que son relevantes".

- ***Recall***: se calcula para cada aplicación y se denomina "sensibilidad o exhaustividad". Se define como "la fracción de elementos relevantes que han sido identificados".

- ***F-measure***: se calcula para cada aplicación y se define como "la media armónica entre la *Precision* y el *Recall*".

La situación ideal es considerada aquella en la que existe una *Precision* y un *Recall* cercanos al 100%, para ello, el parámetro *F-measure* nos ayuda a ponderar estas medidas de forma conjunta.

		Real Clasification		
		True Torrent	True Web	Total
Clustering Clasification	Clas. Torrent	TT	FT	TT+FT
	Clas. Web	FW	TW	FW+TW
	Total	TT+FW	FT+TW	All

Accuracy = TT+TW/All
Precision Torrent = TT/(TT+FT)
Recall Torrent = TT/(TT+FW)
F-measure= 2/((1/precision)+(1/recall))

Precision Web = TW(TW+FW)
Recall Web = TW/(TW+FT)

TT = True Torrent FW = False Web
TW = True Web FT = False Torrent

Figura 13. Ejemplo de cálculo de matriz de confusión para dos aplicaciones

4. Resultados y discusión

Las pruebas realizadas en este proyecto se han basado en intentar clasificar el tráfico generado por dos aplicaciones de distinta naturaleza de forma que la extracción de estadísticos de los flujos actúen como base para detectar dicha diferencia y permitir una clasificación de los flujos veraz.

Dos tipos de tráfico que cuentan con esas características son el tráfico de navegación *Web* y el tráfico *P2P*. El primero se basa en una estructura cliente-servidor en la que encontraremos un número de flujos reducido, con una duración relativamente corta y con una cantidad de datos transmitidos pequeña, representando estos flujos la descarga de páginas web por parte del navegador junto con imágenes o contenido multimedia que éstas puedan contener.

Por otro lado, el tráfico *P2P* define una red en la que todos los nodos actúan como cliente y servidor. Una de las aplicaciones más extendidas de tráfico *P2P* es el intercambio de archivos de gran tamaño en formato de *torrent* a través del protocolo *BitTorrent.* Este protocolo se caracteriza por que los nodos descargan y suben los archivos de forma simultánea, realizando numerosas conexiones con el resto de nodos, descargando partes del archivo de algunos y compartiendo las partes ya descargadas con otros. Este comportamiento implica la existencia de un gran número de flujos existentes en este tipo de tráfico, además de ser flujos de larga duración y con transferencia de un gran volumen de datos.

Tráfico *Web* y tráfico *Torrent* serán, por tanto, las aplicaciones usadas en este proyecto para realizar las pruebas de funcionamiento.

4.1. Descripción de las capturas de tráfico utilizadas

El primer paso para realizar las pruebas consiste en preparar capturas de tráfico de forma controlada de forma que los flujos generados por ambas aplicaciones cuenten con un origen conocido.

Para generar el tráfico web el uso de cualquier navegador de los existentes es suficiente. En estas pruebas se ha hecho uso del navegador *Google Chrome* para generar tráfico web y para generar tráfico torrent se ha elegido entre la gran variedad de clientes *BitTorrent* que existen como *µTorrent, BitTorrent,* o *Transmission*. En este caso se ha elegido *Transmission*.

Mediante *Wireshark* se ha capturado el tráfico de cada aplicación haciendo uso de los filtros propios del programa. En el caso de la navegación web se ha filtrado el tráfico analizado para capturar tráfico HTTP y HTTPS y el filtro usado en *Wireshark* ha sido el siguiente:

tcp.port == 443 || tcp.port == 80

Siendo los puertos 443 y 80 los correspondientes a HTTPS y HTTP, respectivamente.

Para capturar el tráfico *BitTorrent*, se ha tenido en cuenta el puerto bajo el que funciona el cliente elegido. En el caso de *Transmission* la elección del puerto puede ser manual o aleatoria. Por tanto, basta con elegir un puerto determinado como en este caso es 51413 para poder filtrar por dicho puerto el tráfico analizado en Wireshark. Teniendo en cuenta que el protocolo *BitTorrent* se basa en TCP para la transferencia de datos pero también usa UDP para la negociación entre nodos, será necesario tener en cuenta ambos protocolos. El siguiente filtro nos realizará dicha función:

tcp.port == 51413 || udp.port == 51413

Una vez contamos con las capturas de tráfico, estamos listos para realizar el procesado de datos descrito en el capítulo anterior. En las siguientes secciones se detallarán las dos pruebas realizadas para dos aplicaciones, siendo cada una de ellas con distintas capturas de distintas características.

4.1.1. Primera prueba de clasificación de tráfico de dos aplicaciones

Las características de las dos capturas usadas son las siguientes:

- **Captura de *tráfico Torrent***
  - Duración: 4.871 segundos (1 hora y 21 minutos)
  - Número total de paquetes: 4.030.308
  - Número de flujos obtenidos: 16.131
  - Número total de flujos tras aplicar filtro de mínimo 30 paquetes: 1.320

- **Captura de *tráfico Web***
  - Duración: 5.033 segundos (1 hora y 23 minutos)
  - Número total de paquetes: 172.498
  - Número de flujos obtenidos: 2.681
  - Número total de flujos tras aplicar filtro de mínimo 30 paquetes: 656

Como se puede observar, ambas capturas tienen una duración suficientemente larga (de más de 1 hora), y basándonos en el número de paquetes podemos ver las diferencias entre ambos tipos de tráfico, tal y como se comentaba anteriormente: en el caso del tráfico *torrent* es mucho mayor. Por otro lado, en la división en flujos, y con un *timeout* de 10 minutos, el número de flujos también es mayor para el tráfico *torrent*.

Como se explicaba en el capítulo 3, durante el preprocesado de los datos se realiza una criba de los flujos para descartar aquellos que estén formados por menos de 30 paquetes con el fin de usar aquellos que cuenten con un mínimo de paquetes que aporten información relevante al conjunto de datos.

Tras realizar este filtrado, vemos que el 23% de flujos en el caso de *Web* son flujos con más de 30 paquetes, mientras que para *Torrent*, sólo el 8% de ellos cumplen con el requisito. Esto se debe a que aunque el número de flujos que se establecen en una red *BitTorrent* es mucho mayor, muchas de estas conexiones son negociaciones entre nodos y otras pueden representar conexiones entre nodos en las que sólo se intercambian una pequeña parte del archivo, por lo que no superan el límite de 30 paquetes. En tráfico *Web*, sin embargo, aunque el número de flujos iniciales es mucho menor, el intercambio de paquetes entre cliente y servidor es más numeroso en general.

Tras ejecutar el *Script SQL* para la obtención de estadísticos de los flujos, que podemos encontrar en el Anexo B, y tras la ejecución del proceso *RapidMiner*, obtenemos los resultados del análisis PCA y del *clustering* de los flujos.

RapidMiner nos ofrece una gran cantidad de resultados, entre ellos, por ejemplo, podemos observar en la figura 14 que para cubrir el 95% de la varianza de los datos han sido necesario 7 componentes principales, los cuales representan el 96,3%, siendo el resto de componentes, a partir del octavo, descartados.

Component	Standard Deviation	Proportion of Variance	Cumulative Variance
PC 1	2.785	0.399	0.399
PC 2	2.280	0.267	0.666
PC 3	1.595	0.131	0.797
PC 4	1.255	0.081	0.878
PC 5	0.885	0.040	0.918
PC 6	0.715	0.026	0.944
PC 7	0.608	0.019	0.963
PC 8	0.482	0.012	0.975

Figura 14. Proporción de varianza representada por cada componente principal

Por otro lado, podemos ver en la figura 15 que el reparto de los elementos en los clusters sigue una proporción bastante parecida a la realidad.

Cluster Model

```
Cluster 0: 1081 items
Cluster 1: 865 items
Total number of items: 1946
```

Figura 15. Reparto de elementos en los clusters

En la figura 16 se representa una combinación de *clustering* con PCA: podemos ver una representación de los elementos tomando como ejes los dos primeros componentes principales y coloreando los elementos según el clúster al que han sido asignados, siendo los elementos rojos los pertenecientes al clúster 0 y los azules, al clúster 1.

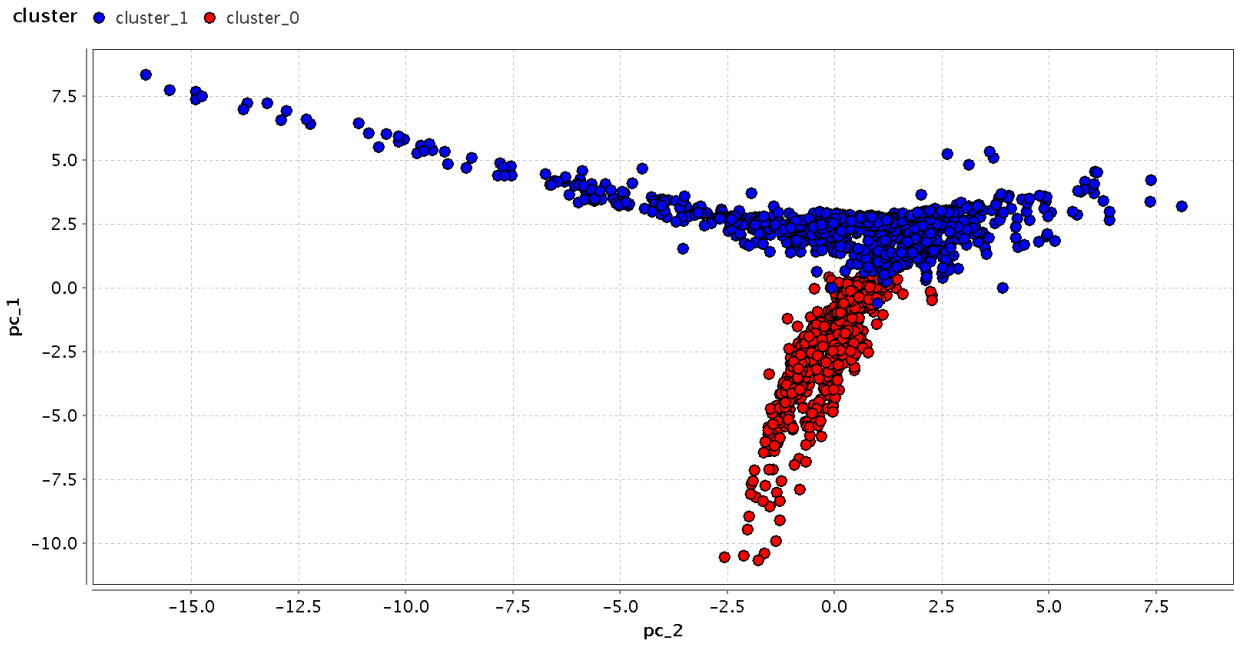

Figura 16. Elementos representados según pca_1 y pca_2 y diferenciados por clusters

Por último, tras la ejecución del proceso *RapidMiner*, contamos con una nueva tabla llamada "*ResultadosClustering*" que podemos cruzar con los datos reales de los flujos alojados en la base de datos para obtener la matriz de confusión. En la figura 17 se muestra la matriz de confusión resultante del clustering de las dos capturas analizadas, junto con los parámetros *Accuracy*, *F-measure*, *Precision* y *Recall*.

		Real Clasification		
		True Torrent	True Web	Total
Clustering Clasification	Clas. Torrent	1011	70	1081
	Clas. Web	280	585	865
	Total	1291	655	1946

Accuracy =	82,01%
F-measure Torrent =	85,24%
F-measure Web =	76,97%
Precision Torrent =	93,52%
Precision Web =	67,63%
Recall Torrent =	78,31%
Recall Web =	89,31%

Figura 17. Matriz de confusión con los resultados obtenidos

Gracias a la matriz de confusión podemos comprobar que el proceso de análisis de datos realizado ha obtenido un *Accuracy* global del 82%.

De los 1.081 flujos que han sido agrupados como "flujos *torrent*", 1.011 eran correctos, lo cual nos da una *Precision* del 93,52%. De los 865 clasificados como *web*, 655 eran, efectivamente, flujos procedentes del tráfico *web*, resultando en una *Precision* del 67,63%. La alta precisión obtenida para el tráfico *torrent* permite afirmar que si un flujo ha sido clasificado como *torrent*, es altamente probable que realmente se trate de un flujo *torrent*, algo que es algo menos probable en el caso de *web*. Se trata de un resultado interesante puesto que si se utiliza el algoritmo para "penalizar" de alguna manera el tráfico P2P, interesa minimizar en la medida de lo posible los "falsos positivos" de manera que no se penalicen injustamente otros tipos de tráfico.

Sin embargo, también se deben observar aquellos elementos que son de una aplicación pero han sido clasificados como la otra. El *Recall* (sensibilidad) nos ayuda a medir este suceso en el que, por ejemplo, de los 1.291 flujos *torrent* reales, 280 han sido clasificados como *web*, dando como resultado un *Recall* del 78,31%, mientras que en el caso de *web*, un 89,31%. En este caso, por lo tanto, el parámetro es de mayor calidad en el caso de *web*, al contrario que ocurría con el caso de la precisión. Esto nos indica que hay proporcionalmente muy pocos flujos que realmente son *web* pero son clasificados por el algoritmo como *torrent*.

Por último, los parámetros *F-measure* nos ayudan a medir *Precision* y *Recall* de forma conjunta para cada una de las aplicaciones.

4.1.2. Segunda prueba de clasificación de tráfico de dos aplicaciones

Esta segunda prueba se ha realizado con un número de flujos mayor. Las características de las dos capturas usadas son las siguientes:

- **Captura de *tráfico Torrent***
  - Duración: 3.567 segundos (59 minutos)
  - Número total de paquetes: 4.632.515
  - Número de flujos obtenidos: 30.051
  - Número total de flujos tras aplicar filtro de mínimo 30 paquetes: 5.043

- **Captura de *tráfico Web***
  - Duración: 2.903 segundos (48 minutos)
  - Número total de paquetes: 516.093
  - Número de flujos obtenidos: 3.245
  - Número total de flujos tras aplicar filtro de mínimo 30 paquetes: 1.163

Como se puede ver, en este caso se va a trabajar con un total de 6.206 flujos, bastantes más que en el caso anterior. Para la preparación de estas pruebas se ha hecho un uso más intensivo tanto de las descargas *torrent* como de la navegación *web*, de ahí a que aunque la duración de las capturas sea menor, cuentan con un número de paquetes mayor. También se puede observar que la reducción del número de flujos al aplicar el mínimo de 30 paquetes se mantiene de la misma forma que en la prueba anterior: el número de flujos *torrent* se reduce mucho más que el de *web*.

En este caso, podemos observar en la figura 18 cómo han sido necesarios 8 componentes principales para alcanzar el 95% de varianza entre los datos.

Component	Standard Deviation	Proportion of Variance	Cumulative Variance
PC 1	2.616	0.298	0.298
PC 2	2.284	0.227	0.524
PC 3	2.066	0.186	0.710
PC 4	1.680	0.123	0.833
PC 5	1.047	0.048	0.880
PC 6	0.862	0.032	0.913
PC 7	0.787	0.027	0.940
PC 8	0.631	0.017	0.957
PC 9	0.587	0.015	0.972

Figura 18. Componentes Principales resultantes de la segunda prueba

Además, podemos observar en la figura 19 el reparto de flujos entre los clusters, siguiendo de igual manera un reparto relativamente similar a la realidad.

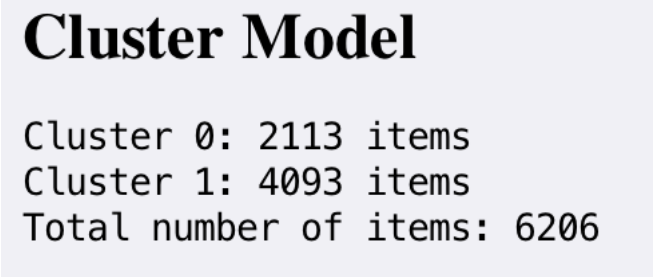

Figura 19. Reparto de flujos entre los clusters en la segunda prueba

Por último, la matriz de confusión presente en la figura 20 nos permite ver los resultados del proceso de clustering entre las dos aplicaciones *torrent* y navegación *web*.

		Real Clasification		
		True Torrent	True Web	Total
Clustering Clasification	Clas. Torrent	3881	212	4093
	Clas. Web	1162	951	2113
	Total	5043	1163	6206

Accuracy =	77,86%
F-measure Torrent =	84,96%
F-measure Web =	58,06%
Precision Torrent =	94,82%
Precision Web =	45,01%
Recall Torrent =	76,96%
Recall Web =	81,77%

Figura 20. Matriz de Confusión de la segunda prueba entre dos aplicaciones

Se puede ver cómo el *Accuracy* global se ha reducido en 5 puntos con respecto a la prueba anterior, bajando desde un 83% hasta un 77%. Esta bajada puede ser causa tanto del aumento del número de flujos, como de las propias características de las capturas realizadas. También se puede observar que parte de esa reducción se ha producido en la clasificación de los flujos *web*, viendo como el parámetro *F-measure Web* ha bajado a un 58%, siendo en la anterior prueba del 77%. En cuanto a los parámetros *Precision* y *Recall*, podemos ver que los correspondientes a *torrent* se han mantenido muy parecidos, apenas reduciéndose en 1 punto, mientras que los relativos a la clasificación web han sufrido una gran reducción, especialmente el parámetro *Precision*, habiendo bajado un 22%.

Es importante mencionar que la relación entre la precisión y la sensibilidad de la clasificación de los flujos observada en la prueba anterior no sólo se mantiene, sino que se acentúa. La precisión con la que los flujos de tráfico *web* han sido clasificados se ha reducido más aún, hasta un 45%, y aunque se siga manteniendo alta, la sensibilidad (*Recall)* también se ha visto reducida hasta un 81%. Por otro lado la precisión de la clasificación de flujos *torrent* se ha mantenido muy alta, en casi un 95%, mientras que el *Recall* está en un 77%, algo menor que en el caso de *web*, tal y como ocurría en la prueba anterior.

4.2. Discusión

En la *primera prueba de clasificación de tráfico de dos aplicaciones* podemos observar que se han obtenido unos resultados satisfactorios. En total, casi 2000 flujos han sido clasificados en dos *clusters* con un grado de precisión (*Accuracy*) del 82%. Se puede observar cómo la extracción de estadísticos y el procesado de éstos por el algoritmo PCA permite caracterizar los flujos de las dos aplicaciones de una manera bastante adecuada, permitiendo que, al aplicar el algoritmo de *clustering* (*k-means*) en el conjunto de datos dimensionado en función de sus Componentes Principales, los flujos contenidos dentro de un mismo clúster son, en su mayoría, provenientes de la misma aplicación.

Se podría plantear un caso hipotético en el que las dos capturas usadas fueran una muestra de tráfico de una red local determinada que sabemos es usada para navegar por Internet y para descargar contenido multimedia a través de *BitTorrent*. Con ello, tras realizar el proceso completo y comprobar que el 55.5% de los flujos pertenecen a un clúster y el 45.5% al otro, se podría realizar un análisis heurístico de un pequeño conjunto de los flujos asociados a cada clúster para determinar a cuál de las dos aplicaciones pertenecen.

Una vez hemos confirmado la procedencia de los flujos de cada cluster, es aceptable afirmar que el 55.5% del tráfico es usado para descargas *torrent* y el 45.5%, para navegación web. Esta afirmación estaría condicionada con el porcentaje de éxito anteriormente comentado, que es del 82%, luego el reparto real en el número de flujos de uno y otro tráfico no sería exactamente el anteriormente comentado, siendo los reales un 66,3% para *torrent* y un 33,7% para web. Aún así, los resultados obtenidos serían útiles para determinar que la descarga de contenido multimedia en esa hipotética red local es el principal consumidor de ancho de banda.

Si se quisiera realizar una mejor distribución de los recursos de dicha red, se podrían tomar medidas como por ejemplo limitar el porcentaje de ancho de banda para dicha actividad a, por ejemplo, un 25% o, por otro lado, si se considera que la descarga de contenido multimedia no es el fin para el que dicha red existe, se podría realizar una prohibición del uso de dichas aplicaciones.

Por otro lado, en la *segunda prueba de clasificación de tráfico de dos aplicaciones* se ha aumentado el número de flujos de una manera significativa, hasta 6.200. Esta segunda prueba ha arrojado unos resultados algo peores que la anterior en varios de los indicadores, pero también suficientemente buenos como para considerarla satisfactoria. El grado de precisión global (*Accuracy*) de clasificación de los flujos alcanzado ha sido del 77%.

Los resultados obtenidos en la segunda prueba comparados con la primera ilustran cómo los pequeños defectos observados en la primera prueba se acentúan al aumentar el número de flujos. La clasificación de tráfico *web* es la más afectada por esto, reduciéndose sus parámetros de *Precision* y *Recall* en varios puntos con respecto a la primera. La clasificación de flujos *torrent*, sin embargo, se mantiene en niveles similares.

La reducción de la precisión global, además del resto de parámetros, puede ser atribuida pues a la presencia de más flujos torrent en proporción a los flujos de *web*. La gran presencia de flujos *torrent* ante flujos *web*, (5000 frente a 1000) hace que el número de flujos *web* que son clasificados como *torrent* de forma errónea por parte del algoritmo aumente. No se debe olvidar que el algoritmo se basa en buscar diferencias entre las características de los flujos y que aunque como ya se ha comentado, el tráfico *torrent* difiere bastante del tráfico *web*, es razonable admitir que la manera concreta de obtener las capturas (los flujos concretos que se capturan) puede influir significativamente en la probabilidad de clasificar adecuadamente cada uno.

5. Conclusiones y trabajos futuros

Con este proyecto se ha buscado el diseño y la implementación de un proceso de análisis y clasificación de tráfico de Internet mediante técnicas de análisis de datos que fueran más allá de la clasificación de los flujos de tráfico a través de valores conocidos (*well-known values*) como pueden ser los puertos TCP o UDP. Para ello, se ha propuesto el uso de técnicas heurísticas en conjunto con técnicas de minería de datos como son el Análisis de Componentes Principales (PCA por sus siglas en inglés), y técnicas de agrupamiento de datos (*clustering*) que agrupen dichos flujos según su aplicación de procedencia. Para ello se ha realizado una preparación y procesado de los datos que ha consistido, en primer lugar, en la generación y captura de tráfico en un entorno controlado que ha permitido contar con una clasificación real de los flujos de dicho tráfico con respecto a la aplicación de la que proceden.

A continuación, se ha realizado una extracción de información de cada uno de estos flujos como direcciones IP origen y destino, tiempo de llegada o la longitud de cada uno de los paquetes que componen dichos flujos.

La información extraída de cada paquete ha sido usada para el cálculo de parámetros estadísticos que ayuden a caracterizar cada flujo. Estos parámetros son los que han sido usados en el Análisis de Componentes Principales, redimensionando el conjunto de datos en función de los componentes principales extraídos.

Haciendo uso de las características matemáticas de dichos componentes principales, se ha aplicado el algoritmo de clustering *k-means* sobre el nuevo conjunto de datos, el cual se encarga de agrupar los flujos en *k* clusters (grupos). Si los parámetros estadísticos extraídos son lo suficientemente representativos, los flujos agrupados en cada cluster pertenecerán en su mayoría a una misma aplicación, permitiendo entonces clasificar el tráfico original.

Una vez desarrollado e implementado todo el proceso, se han realizado varias pruebas para comprobar el funcionamiento de éste. Se han elegido dos aplicaciones de naturaleza suficientemente diferenciada como son la navegación web y la descarga de contenido multimedia a través del protocolo *BitTorrent*.

Mientras que la primera consiste en una arquitectura de red basada en el modelo cliente-servidor, las aplicaciones de descarga de contenido basadas en el protocolo *BitTorrent* siguen un modelo P2P (*Peer to Peer*) en el que cada nodo actúa tanto de cliente como de servidor: la descarga de los archivos se realiza por partes y cuando un nodo cuenta con varias partes de un archivo descargadas, éste las comparte con el resto de la red generado tráfico de subida. Este tipo de tráfico genera, pues, una gran número de flujos entre todos los nodos participantes, además de intercambiarse una gran cantidad de información. Además, estas dos aplicaciones no sólo presentan presumiblemente características muy diferenciadas, sino que además son representativas de usos habituales de las redes IP en entornos empresariales, académicos y residenciales.

Se han detallado los resultados de dos pruebas realizadas con tráfico procedente de las dos aplicaciones navegación web y *torrent*. En la primera se ha realizado una clasificación de casi 2000 flujos de tráfico, obteniendo una una precisión global del 82% en cuanto a flujos asignados a su aplicación correspondiente.

En la segunda, más de 6000 flujos han sido clasificados, obteniendo unos resultados algo más bajos que en la prueba anterior, siendo un 77% la precisión global obtenida.

En general, las pruebas realizadas han arrojado unos resultados que se pueden considerar lo suficientemente buenos para obtener una clasificación de tráfico válida, al menos para estimar el reparto del uso de los recursos de la red que se debe a cada una de las aplicaciones.

Tanto en la primera prueba como en la segunda, los porcentajes de éxito son lo suficientemente altos como para concluir que el proceso de análisis de tráfico mediante análisis estadístico propuesto en este proyecto es válido para aquellos casos en los que se analicen flujos de tráfico de dos aplicaciones con una naturaleza suficientemente diferenciada como son las dos analizadas en las pruebas: la navegación web y el tráfico *torrent* basado en P2P.

En cuanto a trabajos futuros, se propone la ampliación del proceso diseñado para calcular nuevos parámetros estadísticos que ayuden a caracterizar de forma más eficaz el tráfico mediante el algoritmo de Análisis de Componentes Principales. Actualmente, el número de parámetros extraídos es bastante amplio, sin embargo, podría ser posible extraer más información de los paquetes de cada flujo que permita calcular nuevos estadísticos que incrementen la capacidad de caracterización del tráfico durante el proceso de minería.

También se sugiere la realización de pruebas de clasificación añadiendo una tercera aplicación cuya naturaleza sea lo suficientemente distinta a la de las dos tratadas. Aunque la mayoría del tráfico de Internet tiene una naturaleza similar a uno de los dos tipos analizados (web y P2P), existen aplicaciones cuyo tráfico puede diferir lo suficiente como para definir un nuevo “género”. Se plantea entonces como trabajo futuro tener en cuenta esta posibilidad y ampliar el trabajo realizado para que sea capaz de agrupar flujos procedentes de tráfico de red en tres clusters distintos, representando cada uno de ellos un género de tráfico distinto.

6. Referencias bibliográficas

[1] KUMAR, Sailesh; TURNER, Jonathan; WILLIAMS, John. Advanced algorithms for fast and scalable deep packet inspection. En Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems. ACM, 2006. p. 81-92.

[2] MCGREGOR, Anthony, et al. Flow clustering using machine learning techniques. En Passive and Active Network Measurement. Springer Berlin Heidelberg, 2004. p. 205-214.

[3] ANANTAVRASILP, Isara; SCHOLER, Thorsten. Automatic flow classification using machine learning. En Software, Telecommunications and Computer Networks, 2007. SoftCOM 2007. 15th International Conference on. IEEE, 2007. p. 1-6.

[4] LU, Wei; XUE, Ling. A Heuristic-Based Co-clustering Algorithm for the Internet Traffic Classification. En Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on. IEEE, 2014. p. 49-54.

[5] THOTA, Harsha Sai; VEDULA, Vijaya Saradhi; VENKATESH, T. Network Traffic Analysis Using Principal Component Graphs. 2013.

[6] MIRKIN, Boris. Core Concepts in Data Analysis: Summarization, Correlation and Visualization: Summarization, Correlation and Visualization. Springer Science & Business Media, 2011.

[7] Xiaming Chen, Shanghai Jiao Tong Univ, “A simple utility to classify packets into flows”. https://github.com/caesar0301/pkt2flow [Acceso: junio 2015]

[8] Wireshark, Display Filter Reference. https://www.wireshark.org/docs/dfref/ [Acceso: junio 2015]

[9] tshark - Dump and analyze network traffic. https://www.wireshark.org/docs/man-pages/tshark.html [Acceso: junio 2015]

[10] FUKUDA, Kensuke, et al. A pca analysis of daily unwanted traffic. En Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010. p. 377-384.

[11] BRAUCKHOFF, Daniela; SALAMATIAN, Kave; MAY, Martin. Applying PCA for traffic anomaly detection: Problems and solutions. En INFOCOM 2009, IEEE. IEEE, 2009. p. 2866-2870.

[12] RapidMiner en SourceForge, Página Web. http://sourceforge.net/projects/rapidminer/ [Acceso: junio 2015]

[13] Orange, Página Web. http://orange.biolab.si/ [Acceso: junio 2015]

[14] Weka, Página Web. http://www.cs.waikato.ac.nz/ml/weka/ [Acceso: junio 2015]

[15] KNIME, Página Web. https://www.knime.org/ [Acceso: junio 2015]

[16] Wireshark, Pagina Web. https://www.wireshark.org/ [Acceso: junio 2015]

[17] MySQL Community Downloads, Página Web. http://dev.mysql.com/downloads/ [Acceso: junio 2015]

Anexo A: Clase Java ‘CargaBaseDatos’

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class CargaBaseDatos{
	private static Connection connect = null;
	private static Statement statement = null;
	private static PreparedStatement preparedStatement = null;
	private static ResultSet resultSet = null;

	public static void main(String[] args) throws Exception {
		try {
			// This will load the MySQL driver, each DB has its own
driver
			Class.forName("com.mysql.jdbc.Driver");
			// Setup the connection with the DB
			connect = DriverManager

	.getConnection("jdbc:mysql://localhost/PFG_BBDD?"
					+
"user=root&password=PFGDatabase");

			//  Statements  allow  to  issue  SQL  queries  to  the
database
			statement = connect.createStatement();

			String                myDirectoryPath                =
"/Users/luisgil/Desktop/PFG/divisiones";
			File dir = new File(myDirectoryPath);
			File[] directories1 = dir.listFiles(); // 1 folder for
each pcap

	// divided file called

	// X.pcap where X is the aplication name
			for (File directory1 : directories1) {
				if (directory1.isDirectory()) {
					String           directory1name           =
```

```
directory1.getName();
                              String          aplicationName[]          =
directory1name.split("\\.");
                              String aplication = aplicationName[0];

                              preparedStatement = connect
                                          .prepareStatement("insert   into
APLICATIONS              values              (default,              ?)",
PreparedStatement.RETURN_GENERATED_KEYS);
                              preparedStatement.setString(1, aplication);
                              preparedStatement.executeUpdate();
                              ResultSet               rs2               =
preparedStatement.getGeneratedKeys();
                              int id_app = 0;
                              if (rs2.next()) {
                                    id_app = rs2.getInt(1);
                              }
                              preparedStatement.close();

                              System.out.printf("Processing folder %s \n",
directory1.getCanonicalPath());


                              File[]            directories2            =
directory1.listFiles(); // 2 folders:

                              // tcp_syn

                              // and udp
                              for (File directory2 : directories2) {
                                    if (directory2.isDirectory()) {
                                          System.out.printf("Processing
folder %s \n", directory2.getCanonicalPath());

                                          File[]          files          =
directory2.listFiles(); // the actual

                              // files
                                          for (File file : files) {
                                                procesarCSV(file, id_app);
                                          }
                                    }
                              }
                        }
                  }
            } catch (Exception e) {
                  throw e;
            } finally {
                  close();
            }
      }
```

```
	private static void procesarCSV(File child, int id_app)
			throws SQLException {

		String name = child.getName();
		String nombreArchivo[] = name.split("\\.");

		if (nombreArchivo[nombreArchivo.length - 1].equals("csv")) {
			ResultSet rs = statement
					.executeQuery("SELECT COUNT(DISTINCT flowid)
AS idflow_count FROM Packets");
			int id_flow = 0;
			if (rs.next()) {
				id_flow = rs.getInt("idflow_count") + 1;
			}
			BufferedReader br = null;
			String line = "";
			String cvsSplitBy = "%";

			try {
				br = new BufferedReader(new FileReader(child));
				while ((line = br.readLine()) != null) {
					String[] frame = line.split(cvsSplitBy);
					preparedStatement = connect
							.prepareStatement("insert   into
Packets values (default, ?, ?, ?, ? , ?, ?,?,?,?,?)");
					preparedStatement.setString(1, frame[0]);
					preparedStatement.setString(2, frame[1]);
					preparedStatement.setString(3, frame[2]);
					preparedStatement.setString(4, frame[3]);
					preparedStatement.setString(5, frame[4]);
					preparedStatement.setString(6, frame[5]);
					preparedStatement.setString(7, frame[6]);
					if(frame.length > 7){
						preparedStatement.setString(8,
frame[7]);
					}else{
						preparedStatement.setInt(8, 0);
					}
					preparedStatement.setInt(9, id_flow);
					preparedStatement.setString(10, name);

					preparedStatement.executeUpdate();
					preparedStatement.close();
				}
				preparedStatement = connect
						.prepareStatement("insert         into
Clasification values (?, ?)");
				preparedStatement.setInt(1, id_flow);
				preparedStatement.setInt(2, id_app);
				preparedStatement.executeUpdate();
				preparedStatement.close();
			} catch (FileNotFoundException e) {
```

```
                        e.printStackTrace();
                  } catch (IOException e) {
                        e.printStackTrace();
                  } finally {
                        if (br != null) {
                              try {
                                    br.close();
                              } catch (IOException e) {
                                    e.printStackTrace();
                              }
                        }
                  }

            }
      }
      private static void close() {
            try {
                  if (resultSet != null) {
                        resultSet.close();
                  }

                  if (statement != null) {
                        statement.close();
                  }

                  if (connect != null) {
                        connect.close();
                  }
                  System.out.printf("End");

            } catch (Exception e) {

            }
      }

}
```

Anexo B: Script SQL para la extracción de estadísticos de los flujos

```
DROP TABLE IF EXISTS `FlowsData`;
DROP TABLE IF EXISTS `PacketsByDirection`;
DROP TABLE IF EXISTS `PacketsByDirection2`;
DROP TABLE IF EXISTS `PacketsTimesByDirection`;
DROP TABLE IF EXISTS `NodesPerFlowData`;
DROP TABLE IF EXISTS `CombinedData`;
DROP TABLE IF EXISTS `TimesStatistics`;
DROP TABLE IF EXISTS `CombinedStatisticsData`;
DROP TABLE IF EXISTS `SelectionStatisticData`;
DROP TABLE IF EXISTS `FinalTable`;
DROP TABLE IF EXISTS `Packets2`;


SET @minimum_Packets = 30;

CREATE TABLE Packets2 LIKE Packets;
INSERT Packets2 SELECT * FROM Packets;

CREATE TABLE FlowsData
SELECT flowid as FlowID, count(*) as TotalFlowPackets , sum(iplength)
TotalFlowBytes,
max(FrameTimeEpoch) - min(FrameTimeEpoch) as FlowDuration,
IF(count(distinct(ipsrc))=1,'YES','NO') as UniqueDirection
FROM PFG_BBDD.Packets2 group by flowid order by flowid;

CREATE INDEX Index2
ON FlowsData (TotalFlowPackets);
CREATE INDEX Index3
ON FlowsData (flowid);

DELETE FROM FlowsData WHERE TotalFlowPackets < @minimum_Packets;

DELETE FROM Packets2
WHERE flowid NOT in (SELECT DISTINCT flowid FROM FlowsData);


CREATE TABLE NodesPerFlowData
SELECT FlowID, IPSrc, count(*) as Packets, sum(IPLength) as TotalBytes,
max(iplength) as MaxBytes,min(iplength) as MinBytes,avg(iplength) as
AverageBytes,
std(iplength) as BytesStandardDeviation, variance(iplength) as
BytesVariance,MIN(framenumber) as FirstFrame
FROM PFG_BBDD.Packets2 group by ipsrc,flowid order by flowid;

CREATE INDEX Index2
ON NodesPerFlowData (totalbytes);
CREATE INDEX Index3
ON NodesPerFlowData (flowid);

CREATE TABLE PacketsByDirection
SELECT t1.FrameNumber,t1.FrameTimeEpoch,t1.IPSrc,
t1.TCPSrcPort,t1.UDPSrcport,t1.IPDst,t1.TCPDstport,t1.UDPDstport
      ,t1.IPLength,t1.FlowID,
      CASE 1
```

```
    WHEN t1.IPSrc=(SELECT ipsrc FROM PFG_BBDD.NodesPerFlowData
      WHERE (flowid=t1.FlowID AND totalBytes = (SELECT max(totalBytes) FROM
NodesPerFlowData WHERE flowid = t1.FlowID)))
    THEN 'Direction A'
      WHEN t1.IPSrc!=(SELECT ipsrc FROM PFG_BBDD.NodesPerFlowData
      WHERE (flowid=t1.FlowID AND totalBytes = (SELECT max(totalBytes) FROM
NodesPerFlowData WHERE flowid = t1.FlowID)))
    THEN 'Direction B'
    END as Direction
FROM PFG_BBDD.PACKETS2 as t1;

CREATE INDEX Index2
ON PacketsByDirection (direction);
CREATE INDEX Index3
ON PacketsByDirection (flowid);
CREATE INDEX Index4
ON PacketsByDirection (framenumber);

CREATE TABLE PacketsTimesByDirection
 SELECT
      FrameNumber,IPSrc,
      case
      when FlowID <> @currflow or Direction <> @currdir then @diff := 0 else
@diff := FrameTimeEpoch - @epoch
      end as TimeFromLastPacketFromSameDirection
      , @currflow := FlowID AS FlowID, @currdir := Direction as Direction,
@diff as TimeDifference, @epoch := FrameTimeEpoch as FrameTimeEpoch
    FROM
      PacketsByDirection, (select @epoch := 0, @currflow :="", @currdir :=
"", @diff := 0) as tmp
    ORDER BY FlowID, Direction, FrameTimeEpoch;

CREATE TABLE TimesStatistics
SELECT FlowID,IPsrc,Direction,
avg(TimeFromLastPacketFromSameDirection) as TimesAverage,
std(TimeFromLastPacketFromSameDirection) as TimesStandardDeviation,
variance(TimeFromLastPacketFromSameDirection) as TimesVariance
FROM PFG_BBDD.PacketsTimesByDirection group by flowid,ipsrc;

CREATE TABLE CombinedStatisticsData
SELECT t1.FlowID, t1.IPsrc, Direction, TimesAverage, TimesStandardDeviation,
TimesVariance, Packets, TotalBytes,
MaxBytes, MinBytes,AverageBytes,BytesStandardDeviation,BytesVariance
FROM PFG_BBDD.TimesStatistics as t1 INNER JOIN NodesPerFlowData as t2
ON t1.flowid=t2.flowid AND t1.ipsrc=t2.ipsrc;

CREATE INDEX Index2
ON CombinedStatisticsData (direction);
CREATE INDEX Index3
ON CombinedStatisticsData (flowid);

CREATE TABLE SelectionStatisticData
SELECT   t1.FlowID,   t1.IPsrc   as   IPSrcDirectionA,   t1.TimesAverage   as
TimesAverageDirectionA,
t1.TimesStandardDeviation        as        TimesStandardDeviationDirectionA,
t1.TimesVariance     as     TimesVarianceDirectionA,     t1.Packets     as
PacketsDirectionA,
t1.TotalBytes    as    TotalBytesDirectionA,    t1.    AverageBytes    as
AverageBytesDirectionA,
t1.BytesStandardDeviation        as        BytesStandardDeviationDirectionA,
t1.BytesVariance as BytesVarianceDirectionA,
```

Anexo B: Script SQL para la extracción de estadísticos de los flujos

```
t2.IPsrc as IPSrcDirectionB, t2.TimesAverage as TimesAverageDirectionB,
t2.TimesStandardDeviation        as        TimesStandardDeviationDirectionB,
t2.TimesVariance     as     TimesVarianceDirectionB,     t2.Packets     as
PacketsDirectionB,
t2.TotalBytes    as    TotalBytesDirectionB,    t2.    AverageBytes    as
AverageBytesDirectionB,
t2.BytesStandardDeviation        as        BytesStandardDeviationDirectionB,
t2.BytesVariance as BytesVarianceDirectionB
FROM PFG_BBDD.CombinedStatisticsData as t1
left join PFG_BBDD.CombinedStatisticsData
as t2 on (t1.flowid = t2.flowid and t1.direction = 'Direction A' and (SELECT
UniqueDirection FROM FlowsData as t4 where t4.flowid=t1.flowid) = 'NO')
WHERE ((t1.direction != t2.direction)
  OR
(SELECT UniqueDirection FROM FlowsData as t4 where t4.flowid=t1.flowid) =
'YES')
 order by flowid;


CREATE TABLE FinalTable
SELECT t1.FlowID, TotalFlowPackets, TotalFlowBytes,FlowDuration
 ,TimesAverageDirectionA                ,TimesStandardDeviationDirectionA,
TimesVarianceDirectionA
 ,PacketsDirectionA  ,TotalBytesDirectionA   ,AverageBytesDirectionA  ,
 BytesStandardDeviationDirectionA,BytesVarianceDirectionA
 ,TimesAverageDirectionB                 ,TimesStandardDeviationDirectionB
,TimesVarianceDirectionB   ,PacketsDirectionB
,TotalBytesDirectionB,AverageBytesDirectionB
,BytesStandardDeviationDirectionB,BytesVarianceDirectionB
,IFNULL (PacketsDirectionA/PacketsDirectionB,0)  as PacketsRelationAtoB,
IFNULL (TotalBytesDirectionA/TotalBytesDirectionB,0) as BytesRelationAtoB,
IFNULL             (TotalBytesDirectionA/PacketsDirectionA,0)             as
BytesPerPacketDirectionA,
IFNULL             (TotalBytesDirectionB/PacketsDirectionB,0)             as
BytesPerPacketDirectionB
FROM PFG_BBDD.FlowsData as t1 INNER JOIN SelectionStatisticData as t2 ON
t1.flowid=t2.flowid
```

Anexo B: Script SQL para la extracción de estadísticos de los flujos

Anexo C: Código XML de proceso en RapidMiner

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<process version="5.3.013">
  <context>
    <input/>
    <output/>
    <macros/>
  </context>
  <operator    activated="true"    class="process"    compatibility="5.3.013"
expanded="true" name="Process">
    <process expanded="true">
      <operator            activated="true"            class="read_database"
compatibility="5.3.013"  expanded="true"  height="60"  name="Read  Database"
width="90" x="45" y="75">
        <parameter key="connection" value="New connection1"/>
        <parameter key="query" value="SELECT *&#10;FROM `FinalTable`"/>
        <enumeration key="parameters"/>
      </operator>
      <operator  activated="true"  class="set_role"  compatibility="5.3.013"
expanded="true" height="76" name="Set Role" width="90" x="45" y="165">
        <parameter key="attribute_name" value="FlowID"/>
        <parameter key="target_role" value="label"/>
        <list key="set_additional_roles"/>
      </operator>
      <operator        activated="true"        class="replace_missing_values"
compatibility="5.3.013"  expanded="true"  height="94"  name="Replace  Missing
Values" width="90" x="45" y="255">
        <parameter key="default" value="zero"/>
        <list key="columns"/>
      </operator>
      <operator  activated="true"  class="normalize"  compatibility="5.3.013"
expanded="true" height="94" name="Normalize" width="90" x="179" y="255">
        <parameter key="min" value="-3.0"/>
        <parameter key="max" value="3.0"/>
      </operator>
      <operator       activated="true"       class="detect_outlier_distances"
compatibility="5.3.013"  expanded="true"  height="76"  name="Detect  Outlier
(Distances)" width="90" x="179" y="165">
        <parameter key="number_of_neighbors" value="3"/>
        <parameter key="number_of_outliers" value="30"/>
      </operator>
      <operator           activated="true"           class="filter_examples"
compatibility="5.3.013"  expanded="true"  height="76"  name="Filter  Examples"
width="90" x="179" y="30">
        <parameter key="condition_class" value="attribute_value_filter"/>
        <parameter key="parameter_string" value="outlier=false"/>
      </operator>
      <operator    activated="true"    class="principal_component_analysis"
```

```
compatibility="5.3.013" expanded="true" height="94" name="PCA" width="90"
x="313" y="75">
        <parameter key="number_of_components" value="2"/>
      </operator>
      <operator activated="true" class="k_means" compatibility="5.3.013"
expanded="true" height="76" name="Clustering" width="90" x="447" y="30">
        <parameter key="measure_types" value="MixedMeasures"/>
      </operator>
      <operator activated="true" class="write_database"
compatibility="5.3.013" expanded="true" height="60" name="Write Database"
width="90" x="514" y="210">
        <parameter key="connection" value="New connection1"/>
        <parameter key="table_name" value="ResultadosClustering"/>
        <parameter key="overwrite_mode" value="overwrite"/>
      </operator>
      <operator activated="true" class="apply_model" compatibility="5.3.013"
expanded="true" height="76" name="Apply Model" width="90" x="514" y="300">
        <list key="application_parameters"/>
      </operator>
      <connect from_op="Read Database" from_port="output" to_op="Set Role"
to_port="example set input"/>
      <connect from_op="Set Role" from_port="example set output"
to_op="Replace Missing Values" to_port="example set input"/>
      <connect from_op="Replace Missing Values" from_port="example set
output" to_op="Normalize" to_port="example set input"/>
      <connect from_op="Normalize" from_port="example set output"
to_op="Detect Outlier (Distances)" to_port="example set input"/>
      <connect from_op="Detect Outlier (Distances)" from_port="example set
output" to_op="Filter Examples" to_port="example set input"/>
      <connect from_op="Filter Examples" from_port="example set output"
to_op="PCA" to_port="example set input"/>
      <connect from_op="PCA" from_port="example set output"
to_op="Clustering" to_port="example set"/>
      <connect from_op="PCA" from_port="original" to_op="Apply Model"
to_port="unlabelled data"/>
      <connect from_op="PCA" from_port="preprocessing model" to_op="Apply
Model" to_port="model"/>
      <connect from_op="Clustering" from_port="cluster model"
to_port="result 2"/>
      <connect from_op="Clustering" from_port="clustered set" to_op="Write
Database" to_port="input"/>
      <connect from_op="Write Database" from_port="through" to_port="result
3"/>
      <connect from_op="Apply Model" from_port="model" to_port="result 1"/>
      <portSpacing port="source_input 1" spacing="0"/>
      <portSpacing port="sink_result 1" spacing="0"/>
      <portSpacing port="sink_result 2" spacing="0"/>
      <portSpacing port="sink_result 3" spacing="0"/>
      <portSpacing port="sink_result 4" spacing="0"/>
    </process>
  </operator>
</process>
```